CS 341: Algorithms

Lecture 19: Reductions

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2024

Goals for this chapter

- polynomial-time reductions
- **P**, **NP**, **NP**-complete problems
- Cook-Levin: CircuitSAT is **NP**-complete
- many more examples of **NP**-complete problems

Framework

Computational model

So far,

- we used the **word RAM** all the time (CPU has registers that are as large as needed)
- we only counted how many **word operations** we did (unit cost)

This is not well-suited to discuss **P**, **NP**, ...

- use a bit-model instead, where words have **fixed** size (e.g., 1 bit)
  (Cook-Levin's theorem proved for **Turing machines**)
- main difference: should account for the size of integers we represent
  (the **size** of the representation of an integer N is $\lceil \log(N) \rceil + 1 \in \Theta(\log N)$)
- in most cases, runtimes now involve a few extra log terms

does not matter: when talking about **P**, **NP**, ..., we care about polynomial-time-ness, but **not** about precise exponents, log factors, ...

Input size?

when talking about **P**, **NP**, ..., we care about polynomial-time-ness, but **not** about precise exponents, log factors, ...

Example 1: input is an integer M

- size $\textsf{sz}(M) = \lceil \log(M) \rceil + 1 \in \Theta(\log M)$

Example 2: input is an array $A[1..n]$ of integers

- size $S = \sum_i \textsf{sz}(A[i])$
- might as well consider $S' = \boldsymbol{n \max \log(A[i])}$: $T \in S^{O(1)} \iff T \in S'^{O(1)}$
  1. $S \in O(S')$
  2. $S \geq n$ and $S \geq \max \log(A[i])$ so $S \geq \sqrt{S'}$

Input size?

> when talking about **P**, **NP**, ..., we care about polynomial-time-ness, but **not** about precise exponents, log factors, ...

Example 3: graph $G = (V, E)$ with n vertices and m edges

- array $A[1..n]$, each $A[i]$ a list of indices $v_{i,j}$, $j = 1, \ldots, \text{degree}(i)$
- size $S = n + \sum_{i,j} \textbf{sz}(v_{i,j})$
- might as well consider $S' = n + m$

Example 4: directed graph $G = (V, E)$ with n vertices and m edges, with integer weights w:

- array A of size n
- each $A[i]$ a list of pairs $(v_{i,j}, w_{i,j})$, $j = 1, \ldots, \text{out-degree}(i)$
- size $S = n + \sum_{i,j} \textbf{sz}(v_{i,j}) + \textbf{sz}(w_{i,j})$
- might as well consider $S' = n + m \max \log(w)$

What problems do we consider?

Definition.

- a **decision problem** is a problem to which the answer is **yes** or **no**
- write $x \in \textsc{Prob}$ if x is a yes-instance

formally, $\textsc{Prob}$ is a language (a set of strings over e.g. $\{0,1\}$)

Examples

- is graph G a tree? $G \in \textsc{Tree}$
- is graph G colorable with 3 colors? $G \in 3\text{-}\textsc{Colorable}$

Non-examples

- what is the maximum flow through this graph?
- find an assignment of variables that makes a boolean formula true

Optimization vs decision

Optimization problems

- find the maximal flow value in G
- find a minimal spanning tree in G
- optimize a linear function ...

Decision versions of optimization problems:

- given G and K, is there a flow of value $\geq K$?
- given G and K, is there a spanning tree of weight $\leq K$?
- etc.

Remark

- optimization problem solvable in polynomial time $\implies$ decision version solvable in polynomial time
- converse true if the optimum is an integer that fits into a polynomial number of bits

Reductions

Definition

formalizes the idea that you can use **subroutines** to solve new problems.

Key idea:

- if you can solve a problem Prob2 in polynomial time,
- you may use it to solve Prob1 in polynomial time.

Definition

Prob1 can be **polynomial-time reduced** to Prob2 if

- there exists an algorithm C that runs in polynomial time,
- such that $x \in$ Prob1 if and only if $C(x) \in$ Prob2.

Notation: Prob1 $\leq_P$ Prob2.

Remark: also called Karp reductions. Alternative: use Prob2 as an oracle, allowing multiple calls (Cook reductions).

Complexity

Assume

- $\boldsymbol{C}$ runs in time $\boldsymbol{c(n)}$, $n = \text{size(input)}$
  in particular, the output has size at most $c(n)$
- we have an algorithm $\boldsymbol{A_2}$ that solves Prob2 in time $\boldsymbol{a(m)}$, $m = \text{size(input)}$

Consequence

- we get algorithm $\boldsymbol{A_1}$ that solves Prob1 in time $\boldsymbol{c(n) + a(c(n))}$
  (because size of $C(x) \leq c(n)$)
- so polynomial time for Prob2 $\implies$ polynomial time for Prob1

Contrapositive

no polynomial time algorithm for Prob1 $\implies$
no polynomial time algorithm for Prob2

Examples

PROB1	PROB2
subset sum	decision version of 0/1 knapsack (is there a choice of items with value $\geq K$?)
longest increasing subsequence (decision version)	longest common subsequence (decision version)
vertex-disjoint paths (decision version)	edge-disjoint paths (decision version)

(all reductions take polynomial time)

Some graph problems

IndependentSet

- given a graph G and K, is there an independent set of size **at least** K in G?

  independent set: vertices S with $\{u, v\}$ **not an** edge for all u, v in S

VertexCover

- given a graph G and K, is there a vertex cover of size **at most** K in G?

  vertex cover: vertices S s.t. any edge has an extremity in S

Clique

- given a graph G and K, is there a clique of size **at least** K in G?

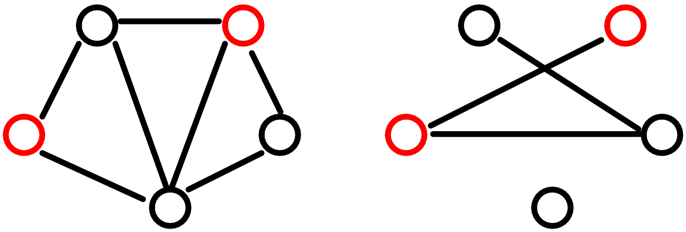
  clique: vertices S with $\{u, v\}$ edge for all u, v in S $(u \neq v)$

Some easy reductions

Let $\bar{G} = (S, \bar{E})$ be the complement graph of G: $e \in E \iff e \notin \bar{E}$

Claim 1

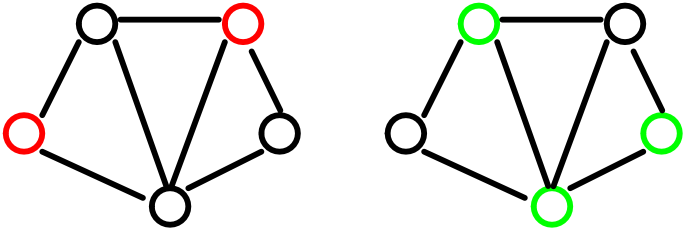
S is an independent set in G **iff** S is a clique in $\bar{G}$



Some easy reductions

Claim 2

S is an independent set in G **iff** $V - S$ is vertex cover in G



Some easy reductions

Claims give

- $\textsc{IndependentSet} \leq_P \textsc{Clique} \leq_P \textsc{IndependentSet}$
- $\textsc{IndependentSet} \leq_P \textsc{VertexCover} \leq_P \textsc{IndependentSet}$

Transitivity: if A $\leq_P$ B and B $\leq_P$ C, then A $\leq_P$ C

Consequence

$\textsc{IndependentSet} \leq_P \textsc{VertexCover} \leq_P \textsc{Clique} \leq_P \textsc{IndependentSet}$

(they are **polynomial-time equivalent**)

Hamiltonian paths and cycles

HamiltonianPath

- given a (symmetric) graph G with n vertices, is there a path $v_1, \ldots, v_n$ that visits all vertices?

HamiltonianCycle

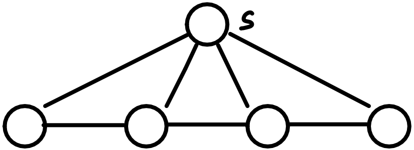
- given a (symmetric) graph G with n vertices, is there a cycle $v_1, \ldots, v_n, v_1$ that visits all vertices?

Remark:

- if there is a Hamiltonian cycle, there is a Hamiltonian path
- but converse may not hold

HamiltonianPath $\leq_P$ HamiltonianCycle

Given G, create G' by adding a **new vertex** s connected to all other vertices



Claim

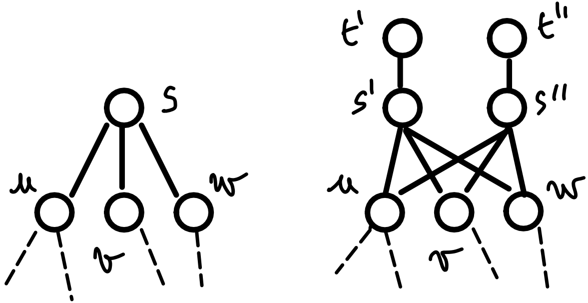
$G \in \textsc{HamiltonianPath} \iff G' \in \textsc{HamiltonianCycle}$

- $v_1, \ldots, v_n$ Hamiltonian path in $G \implies s, v_1, \ldots, v_n, s$ Hamiltonian cycle in G'
- if there is a Hamiltonian cycle in G', we can write it $s, v_1, \ldots, v_n, s$
  then $v_1, \ldots, v_n$ Hamiltonian path in G

Remark: reduction takes polynomial time

HamiltonianCycle $\leq_P$ HamiltonianPath

Given G, create G' by choosing **one vertex** s and using a gadget:



Remark: reduction in polynomial time.

HamiltonianCycle $\leq_P$ HamiltonianPath

Claim

$G \in \textsc{HamiltonianCycle} \iff G' \in \textsc{HamiltonianPath}$

- if there is a Hamiltonian cycle in G, we can write it $s, u, \ldots, w, s$
  $\implies t', s', u, \ldots, s'', t''$ Hamiltonian path in G'
- if there is a Hamiltonian path in G', we can write it $t', s', u, \ldots, w, s'', t''$ or $t'', s'', u, \ldots, w, s', t'$
  $\implies s, u, \ldots, s$ Hamiltonian cycle in G