Complexity Theory

Eric Price

UT Austin

CS 331, Spring 2020 Coronavirus Edition

Note on Reductions

- For an *algorithm*: want to solve NEW. Know how to solve OLD. Reduce NEW to OLD:

Note on Reductions

- For an *algorithm*: want to solve NEW. Know how to solve OLD. Reduce NEW to OLD:
  - Reduce from "Finding largest square in polytope" to LP.

Note on Reductions

- For an *algorithm*: want to solve NEW. Know how to solve OLD. Reduce NEW to OLD:
  - Reduce from “Finding largest square in polytope” to LP.
  - Reduce from “Tile a region with dominos” to network flow.

Note on Reductions

- For an *algorithm*: want to solve NEW. Know how to solve OLD. Reduce NEW to OLD:
  - Reduce from “Finding largest square in polytope” to LP.
  - Reduce from “Tile a region with dominos” to network flow.
  - Reduce from “Solve a word ladder” to shortest path.

Note on Reductions

- For an *algorithm*: want to solve NEW. Know how to solve OLD. Reduce NEW to OLD:
  - Reduce from "Finding largest square in polytope" to LP.
  - Reduce from "Tile a region with dominos" to network flow.
  - Reduce from "Solve a word ladder" to shortest path.
- For a *lower bound*: want to show NEW is hard. Know that OLD is hard. Reduce OLD to NEW:

Note on Reductions

- For an *algorithm*: want to solve NEW. Know how to solve OLD. Reduce NEW to OLD:
  - Reduce from "Finding largest square in polytope" to LP.
  - Reduce from "Tile a region with dominos" to network flow.
  - Reduce from "Solve a word ladder" to shortest path.
- For a *lower bound*: want to show NEW is hard. Know that OLD is hard. Reduce OLD to NEW:
  - Reduce SAT to 3SAT

Note on Reductions

- For an *algorithm*: want to solve NEW. Know how to solve OLD. Reduce NEW to OLD:
  - Reduce from "Finding largest square in polytope" to LP.
  - Reduce from "Tile a region with dominos" to network flow.
  - Reduce from "Solve a word ladder" to shortest path.
- For a *lower bound*: want to show NEW is hard. Know that OLD is hard. Reduce OLD to NEW:
  - Reduce SAT to 3SAT
  - Reduce 3SAT to max-independent set

Note on Reductions

- For an *algorithm*: want to solve NEW. Know how to solve OLD. Reduce NEW to OLD:
  - Reduce from "Finding largest square in polytope" to LP.
  - Reduce from "Tile a region with dominos" to network flow.
  - Reduce from "Solve a word ladder" to shortest path.
- For a *lower bound*: want to show NEW is hard. Know that OLD is hard. Reduce OLD to NEW:
  - Reduce SAT to 3SAT
  - Reduce 3SAT to max-independent set
- If OLD is hard, and you could solve OLD by solving NEW, then NEW must be hard as well.

Models of computation

- Mentioned Word-RAM model early in semester

Models of computation

- Mentioned Word-RAM model early in semester
- Turing machines: tape, heads, etc.

Models of computation

- Mentioned Word-RAM model early in semester
- Turing machines: tape, heads, etc.
- Lambda calculus: church numerals, etc.

Models of computation

- Mentioned Word-RAM model early in semester
- Turing machines: tape, heads, etc.
- Lambda calculus: church numerals, etc.

Theorem

The set of functions that Turing machines can compute is exactly the same as what Lambda calculus can.

Models of computation

- Mentioned Word-RAM model early in semester
- Turing machines: tape, heads, etc.
- Lambda calculus: church numerals, etc.

Theorem

The set of functions that Turing machines can compute is exactly the same as what Lambda calculus can.

Conjecture (Church-Turing Thesis)

Turing machines can compute anything that is computable by any physical process.

Church-Turing Thesis

Conjecture (Church-Turing Thesis)

Turing machines can compute anything that is computable by any physical process.

Church-Turing Thesis

Conjecture (Church-Turing Thesis)

Turing machines can compute anything that is computable by any physical process.

Conjecture (*Extended* Church-Turing Thesis)

Turing machines can compute in polynomial time anything that is computable by any "realistic" physical process.

Church-Turing Thesis

Conjecture (Church-Turing Thesis)

Turing machines can compute anything that is computable by any physical process.

Conjecture (*Extended* Church-Turing Thesis)

Quantum Turing machines can compute in polynomial time anything that is computable by any "realistic" physical process.

Formal(ish) Definitions

Definition (Language)

A "problem" is also referred to as a "language" $L \subseteq \{0,1\}^*$ consisting of YES inputs. An input $x \in \{0,1\}^*$ is "YES" if, and only if, $x \in L$.

Formal(ish) Definitions

Definition (Language)

A "problem" is also referred to as a "language" $L \subseteq \{0,1\}^*$ consisting of YES inputs. An input $x \in \{0,1\}^*$ is "YES" if, and only if, $x \in L$.

Definition (P)

A language L is in P iff there exists a poly-time algorithm $\mathcal{A}$ such that, for all x, $\mathcal{A}(x) = 1$ if and only if $x \in L$.

Formal(ish) Definitions

Definition (Language)

A "problem" is also referred to as a "language" $L \subseteq \{0,1\}^*$ consisting of YES inputs. An input $x \in \{0,1\}^*$ is "YES" if, and only if, $x \in L$.

Definition (P)

A language L is in P iff there exists a poly-time algorithm $\mathcal{A}$ such that, for all x, $\mathcal{A}(x) = 1$ if and only if $x \in L$.

Definition (NP)

A language L is in NP iff there exists a poly-time algorithm $\mathcal{V}$ such that:

1. For all $x \in L$, $\exists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.
2. For all $x \notin L$, $\nexists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.

Formal(ish) Definitions

Definition (Language)

A "problem" is also referred to as a "language" $L \subseteq \{0,1\}^*$ consisting of YES inputs. An input $x \in \{0,1\}^*$ is "YES" if, and only if, $x \in L$.

Definition (P)

A language L is in P iff there exists a poly-time algorithm $\mathcal{A}$ such that, for all x, $\mathcal{A}(x) = 1$ if and only if $x \in L$.

Definition (NP)

A language L is in NP iff there exists a poly-time algorithm $\mathcal{V}$ such that:

1. For all $x \in L$, $\exists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.
2. For all $x \notin L$, $\nexists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.

$P \subseteq NP$: $\mathcal{V}(x,p) := \mathcal{A}(x)$.

Cook-Levin theorem

Definition (NP)

A language L is in NP iff there exists a poly-time algorithm $\mathcal{V}$ such that:

1. For all $x \in L$, $\exists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.
2. For all $x \notin L$, $\nexists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.

- Want to show CircuitSAT is NP-hard.

Cook-Levin theorem

Definition (NP)

A language L is in NP iff there exists a poly-time algorithm $\mathcal{V}$ such that:

1. For all $x \in L$, $\exists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.
2. For all $x \notin L$, $\nexists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.

- Want to show CircuitSAT is NP-hard.
- Goal: reduce from *any* NP problem to CircuitSAT.

Cook-Levin theorem

Definition (NP)

A language L is in NP iff there exists a poly-time algorithm $\mathcal{V}$ such that:

1. For all $x \in L$, $\exists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.
2. For all $x \notin L$, $\nexists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.

- Want to show CircuitSAT is NP-hard.
- Goal: reduce from *any* NP problem to CircuitSAT.
- Imagine that "poly-time algorithm $\mathcal{V}$" were "poly-size circuit $\overline{\mathcal{V}}$":

Cook-Levin theorem

Definition (NP)

A language L is in NP iff there exists a poly-time algorithm $\mathcal{V}$ such that:

1. For all $x \in L$, $\exists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.
2. For all $x \notin L$, $\nexists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.

- Want to show CircuitSAT is NP-hard.
- Goal: reduce from *any* NP problem to CircuitSAT.
- Imagine that "poly-time algorithm $\mathcal{V}$" were "poly-size circuit $\overline{\mathcal{V}}$":
  - Then "is $x \in L$" is the same as $\exists p : \overline{\mathcal{V}}(x,p) = 1$

Cook-Levin theorem

Definition (NP)

A language L is in NP iff there exists a poly-time algorithm $\mathcal{V}$ such that:

1. For all $x \in L$, $\exists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.
2. For all $x \notin L$, $\nexists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.

- Want to show CircuitSAT is NP-hard.
- Goal: reduce from *any* NP problem to CircuitSAT.
- Imagine that "poly-time algorithm $\mathcal{V}$" were "poly-size circuit $\overline{\mathcal{V}}$":
  - Then "is $x \in L$" is the same as $\exists p : \overline{\mathcal{V}}(x,p) = 1$
  - Which is just CircuitSAT on $\overline{\mathcal{V}}(x,\cdot)$

Cook-Levin theorem

Definition (NP)

A language L is in NP iff there exists a poly-time algorithm $\mathcal{V}$ such that:

1. For all $x \in L$, $\exists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.
2. For all $x \notin L$, $\nexists p \in \{0,1\}^*$ of poly. length s.t. $\mathcal{V}(x,p) = 1$.

- Want to show CircuitSAT is NP-hard.
- Goal: reduce from *any* NP problem to CircuitSAT.
- Imagine that "poly-time algorithm $\mathcal{V}$" were "poly-size circuit $\overline{\mathcal{V}}$":
  - Then "is $x \in L$" is the same as $\exists p : \overline{\mathcal{V}}(x,p) = 1$
  - Which is just CircuitSAT on $\overline{\mathcal{V}}(x,\cdot)$
- So we just need to transform the Turing machine into an equivalent circuit of polynomial size.

Cook-Levin theorem

Reducing Turing machine to SAT by unrolling across time

Variables

$$L_{i,j} := \text{Machine at tape position } j \text{ at time } i$$
$$Q_{i,j} := \text{Machine in state } j \text{ at time } i$$
$$T_{i,j,k} := \text{Tape at position } j \text{ at time } i \text{ has value } k$$

- Polynomial time, states, values $\implies$ polynomially many vars.

Transition rules

If $L_{i,j} \cap T_{i,j,k} \cap Q_{i,\ell}$ then machine moves based on reading k in state ℓ:

$$L_{i+1,t} = 1 \text{ if } t = g(k,\ell) \text{ else } 0 \qquad \text{Move left/right}$$
$$Q_{i+1,t} = 1 \text{ if } t = f(k,\ell) \text{ else } 0 \qquad \text{Change state}$$
$$T_{i+1,j,t} = 1 \text{ if } t = h(k,\ell) \text{ else } 0 \qquad \text{Write new char.}$$

Example: $g(k,\ell) = \ell + 1$ if, when reading k in state ℓ, you move right.

Cook-Levin theorem

Reducing Turing machine to SAT by unrolling across time

Variables

$$L_{i,j} := \text{Machine at tape position } j \text{ at time } i$$
$$Q_{i,j} := \text{Machine in state } j \text{ at time } i$$
$$T_{i,j,k} := \text{Tape at position } j \text{ at time } i \text{ has value } k$$

- Polynomial time, states, values $\implies$ polynomially many vars.

Transition rules

- Add a few more rules (e.g., machine in only one state at each time).

Cook-Levin theorem

Reducing Turing machine to SAT by unrolling across time

Variables

$$L_{i,j} := \text{Machine at tape position } j \text{ at time } i$$
$$Q_{i,j} := \text{Machine in state } j \text{ at time } i$$
$$T_{i,j,k} := \text{Tape at position } j \text{ at time } i \text{ has value } k$$

- Polynomial time, states, values $\implies$ polynomially many vars.

Transition rules

- Add a few more rules (e.g., machine in only one state at each time).
- Output is what's written when you halt.

Cook-Levin theorem

Reducing Turing machine to SAT by unrolling across time

Variables

$$L_{i,j} := \text{Machine at tape position } j \text{ at time } i$$
$$Q_{i,j} := \text{Machine in state } j \text{ at time } i$$
$$T_{i,j,k} := \text{Tape at position } j \text{ at time } i \text{ has value } k$$

- Polynomial time, states, values $\implies$ polynomially many vars.

Transition rules

- Add a few more rules (e.g., machine in only one state at each time).
- Output is what's written when you halt.
- Turing machine outputs YES if and only if the circuit outputs YES.

Cook-Levin theorem

Reducing Turing machine to SAT by unrolling across time

Variables

$$L_{i,j} := \text{Machine at tape position } j \text{ at time } i$$
$$Q_{i,j} := \text{Machine in state } j \text{ at time } i$$
$$T_{i,j,k} := \text{Tape at position } j \text{ at time } i \text{ has value } k$$

- Polynomial time, states, values $\implies$ polynomially many vars.

Transition rules

- Add a few more rules (e.g., machine in only one state at each time).
- Output is what's written when you halt.
- Turing machine outputs YES if and only if the circuit outputs YES.
- Q for NP verifier: does there exist an initial input (= tape state) such that output is YES?

Kinds of reduction

- Suppose we want to show B is hard by reducing A to B

Kinds of reduction

- Suppose we want to show B is hard by reducing A to B
- Cook reduction: more powerful

Kinds of reduction

- Suppose we want to show B is hard by reducing A to B
- Cook reduction: more powerful
  - Show that, given an oracle for B, can solve A in poly time

Kinds of reduction

- Suppose we want to show B is hard by reducing A to B
- Cook reduction: more powerful
  - Show that, given an oracle for B, can solve A in poly time
  - Can call B many times, do intermediate processing, etc.

Kinds of reduction

- Suppose we want to show B is hard by reducing A to B
- Cook reduction: more powerful
  - Show that, given an oracle for B, can solve A in poly time
  - Can call B many times, do intermediate processing, etc.
  - If $B \in P$ then $A \in P$.

Kinds of reduction

- Suppose we want to show B is hard by reducing A to B
- Cook reduction: more powerful
  - Show that, given an oracle for B, can solve A in poly time
  - Can call B many times, do intermediate processing, etc.
  - If $B \in P$ then $A \in P$.
- Karp reduction: simpler

Kinds of reduction

- Suppose we want to show B is hard by reducing A to B
- Cook reduction: more powerful
  - Show that, given an oracle for B, can solve A in poly time
  - Can call B many times, do intermediate processing, etc.
  - If $B \in P$ then $A \in P$.
- Karp reduction: simpler
  - Transform any instance x of A into a (very specific) instance x' of B (of polynomial size).

Kinds of reduction

- Suppose we want to show B is hard by reducing A to B
- Cook reduction: more powerful
  - Show that, given an oracle for B, can solve A in poly time
  - Can call B many times, do intermediate processing, etc.
  - If $B \in P$ then $A \in P$.
- Karp reduction: simpler
  - Transform any instance x of A into a (very specific) instance x' of B (of polynomial size).
  - Such that x is YES for A if, and only if, x' is YES for B.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
2. Show how to transform any certificate for x into a certificate for y.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
2. Show how to transform any certificate for x into a certificate for y.
   - (Certificate: satisfying assignment / max independent set / Hamiltonian cycle / other short "proof" of YES.)

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
2. Show how to transform any certificate for x into a certificate for y.
   - (Certificate: satisfying assignment / max independent set / Hamiltonian cycle / other short "proof" of YES.)
   - Hence if x is YES, y must be YES.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
2. Show how to transform any certificate for x into a certificate for y.
   - (Certificate: satisfying assignment / max independent set / Hamiltonian cycle / other short "proof" of YES.)
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
2. Show how to transform any certificate for x into a certificate for y.
   - (Certificate: satisfying assignment / max independent set / Hamiltonian cycle / other short "proof" of YES.)
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.
   - Hence if y is YES, x must be YES.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
2. Show how to transform any certificate for x into a certificate for y.
   - (Certificate: satisfying assignment / max independent set / Hamiltonian cycle / other short "proof" of YES.)
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.
   - Hence if y is YES, x must be YES.
4. To prove *NP-completeness*: show $B \in NP$ by showing that *arbitrary* instances of B have certificates.

Karp's 21 NP-complete problems

- Every NP problem reduces to Circuit-SAT
- Circuit-SAT reduces to SAT
- SAT reduces to 3-SAT
- 3-SAT reduces to independent set
  - Independent set reduces to vertex cover
    - Vertex cover reduces to directed Hamiltonian cycle
    - Directed Hamiltonian cycle reduces to undirected hamiltonian cycle
- 3-SAT reduces to graph coloring
  - Chromatic number reduces to exact cover
    - Exact cover reduces to subset sum.

Subset Sum is NP-hard

Definition (Subset Sum)

Given $a_1, a_2, \ldots, a_n > 0$—each represented with poly(n) bits—and a number T, does there exist $S \subseteq [n]$ such that

$$\sum_{i \in S} a_i = T?$$

Subset Sum is NP-hard

Definition (Subset Sum)

Given $a_1, a_2, \ldots, a_n > 0$—each represented with poly(n) bits—and a number T, does there exist $S \subseteq [n]$ such that

$$\sum_{i \in S} a_i = T?$$

- Reduce from vertex cover.

Subset Sum is NP-hard

Definition (Subset Sum)

Given $a_1, a_2, \ldots, a_n > 0$—each represented with poly(n) bits—and a number T, does there exist $S \subseteq [n]$ such that

$$\sum_{i \in S} a_i = T?$$

- Reduce from vertex cover.
- Assign the edges $e \in E$ numbers in $0, \ldots, |E| - 1$.

Subset Sum is NP-hard

Definition (Subset Sum)

Given $a_1, a_2, \ldots, a_n > 0$—each represented with poly(n) bits—and a number T, does there exist $S \subseteq [n]$ such that

$$\sum_{i \in S} a_i = T?$$

- Reduce from vertex cover.
- Assign the edges $e \in E$ numbers in $0, \ldots, |E| - 1$.
- Two kinds of numbers: b_e and a_v, both written in base 4.

Subset Sum is NP-hard

Definition (Subset Sum)

Given $a_1, a_2, \ldots, a_n > 0$—each represented with poly(n) bits—and a number T, does there exist $S \subseteq [n]$ such that

$$\sum_{i \in S} a_i = T?$$

- Reduce from vertex cover.
- Assign the edges $e \in E$ numbers in $0, \ldots, |E| - 1$.
- Two kinds of numbers: b_e and a_v, both written in base 4.
  - $b_e = 000100000_4$, with a 1 at position e.

Subset Sum is NP-hard

Definition (Subset Sum)

Given $a_1, a_2, \ldots, a_n > 0$—each represented with poly(n) bits—and a number T, does there exist $S \subseteq [n]$ such that

$$\sum_{i \in S} a_i = T?$$

- Reduce from vertex cover.
- Assign the edges $e \in E$ numbers in $0, \ldots, |E| - 1$.
- Two kinds of numbers: b_e and a_v, both written in base 4.
  - $b_e = 000100000_4$, with a 1 at position e.
  - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.

Subset Sum is NP-hard

Definition (Subset Sum)

Given $a_1, a_2, \ldots, a_n > 0$—each represented with poly(n) bits—and a number T, does there exist $S \subseteq [n]$ such that

$$\sum_{i \in S} a_i = T?$$

- Reduce from vertex cover.
- Assign the edges $e \in E$ numbers in $0, \ldots, |E| - 1$.
- Two kinds of numbers: b_e and a_v, both written in base 4.
  - $b_e = 000100000_4$, with a 1 at position e.
  - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
- For any vertex cover S of size k,

$$\sum_{v \in S} a_v = k4^{|E|} + \sum_{e} 4^e \cdot \begin{cases} 1 & \text{if } |e \cap S| = 1 \\ 2 & \text{if } |e \cap S| = 2 \end{cases}$$

Subset Sum is NP-hard

Definition (Subset Sum)

Given $a_1, a_2, \ldots, a_n > 0$—each represented with poly(n) bits—and a number T, does there exist $S \subseteq [n]$ such that

$$\sum_{i \in S} a_i = T?$$

- Reduce from vertex cover.
- Assign the edges $e \in E$ numbers in $0, \ldots, |E| - 1$.
- Two kinds of numbers: b_e and a_v, both written in base 4.
  - $b_e = 000100000_4$, with a 1 at position e.
  - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
- For any vertex cover S of size k,

$$\sum_{v \in S} a_v = k4^{|E|} + \sum_e 4^e \cdot \begin{cases} 1 & \text{if } |e \cap S| = 1 \\ 2 & \text{if } |e \cap S| = 2 \end{cases}$$

- Hence $T = k4^{|E|} + 22222222222_4$ is possible.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
   - $b_e = 000100000_4$, with a 1 at position e.
   - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
   - $T = k4^{|E|} + \sum_e 2 \cdot 4^e$

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
   - $b_e = 000100000_4$, with a 1 at position e.
   - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
   - $T = k4^{|E|} + \sum_e 2 \cdot 4^e$
2. Show how to transform any certificate for x into a certificate for y.
   - Done: given cover S, take a_v for $v \in S$ and some b_e as necessary.
   - Hence if x is YES, y must be YES.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
   - $b_e = 000100000_4$, with a 1 at position e.
   - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
   - $T = k4^{|E|} + \sum_e 2 \cdot 4^e$
2. Show how to transform any certificate for x into a certificate for y.
   - Done: given cover S, take a_v for $v \in S$ and some b_e as necessary.
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
   - $b_e = 000100000_4$, with a 1 at position e.
   - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
   - $T = k4^{|E|} + \sum_e 2 \cdot 4^e$
2. Show how to transform any certificate for x into a certificate for y.
   - Done: given cover S, take a_v for $v \in S$ and some b_e as necessary.
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.
   - Certificate for y is a set of S_V and S_E with

$$\sum_{v \in S_V} a_v + \sum_{e \in S_E} b_e = T = k4^{|E|} + \sum_e 2 \cdot 4^e.$$

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
   - $b_e = 000100000_4$, with a 1 at position e.
   - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
   - $T = k4^{|E|} + \sum_e 2 \cdot 4^e$
2. Show how to transform any certificate for x into a certificate for y.
   - Done: given cover S, take a_v for $v \in S$ and some b_e as necessary.
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.
   - Certificate for y is a set of S_V and S_E with

$$\sum_{v \in S_V} a_v + \sum_{e \in S_E} b_e = T = k4^{|E|} + \sum_e 2 \cdot 4^e.$$

   - In base 4, the $e = (u, v)$th digit appears in a_u, a_v, and b_e.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
   - $b_e = 000100000_4$, with a 1 at position e.
   - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
   - $T = k4^{|E|} + \sum_e 2 \cdot 4^e$
2. Show how to transform any certificate for x into a certificate for y.
   - Done: given cover S, take a_v for $v \in S$ and some b_e as necessary.
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.
   - Certificate for y is a set of S_V and S_E with

$$\sum_{v \in S_V} a_v + \sum_{e \in S_E} b_e = T = k4^{|E|} + \sum_e 2 \cdot 4^e.$$

   - In base 4, the $e = (u, v)$th digit appears in a_u, a_v, and b_e.
   - Hence no overflows, and one of a_u or a_v must be picked for each $e \in E$.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
   - $b_e = 000100000_4$, with a 1 at position e.
   - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
   - $T = k4^{|E|} + \sum_e 2 \cdot 4^e$
2. Show how to transform any certificate for x into a certificate for y.
   - Done: given cover S, take a_v for $v \in S$ and some b_e as necessary.
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.
   - Certificate for y is a set of S_V and S_E with

$$\sum_{v \in S_V} a_v + \sum_{e \in S_E} b_e = T = k4^{|E|} + \sum_e 2 \cdot 4^e.$$

   - In base 4, the $e = (u, v)$th digit appears in a_u, a_v, and b_e.
   - Hence no overflows, and one of a_u or a_v must be picked for each $e \in E$.
   - Exactly k of the a_u picked so $4^{|E|}$ term matches.

Recipe for Karp reductions to prove NP-hardness

1. Transform an *arbitrary* instance x of A into a *very specific* instance y of B (of polynomial size).
   - $b_e = 000100000_4$, with a 1 at position e.
   - $a_v = 10100100010010_4$, with a 1 at position e if $v \in e$, and another 1 at position $|E|$.
   - $T = k4^{|E|} + \sum_e 2 \cdot 4^e$
2. Show how to transform any certificate for x into a certificate for y.
   - Done: given cover S, take a_v for $v \in S$ and some b_e as necessary.
   - Hence if x is YES, y must be YES.
3. Show how to transform any certificate for y into a certificate for x.
   - Certificate for y is a set of S_V and S_E with

$$\sum_{v \in S_V} a_v + \sum_{e \in S_E} b_e = T = k4^{|E|} + \sum_e 2 \cdot 4^e.$$

   - In base 4, the $e = (u, v)$th digit appears in a_u, a_v, and b_e.
   - Hence no overflows, and one of a_u or a_v must be picked for each $e \in E$.
   - Exactly k of the a_u picked so $4^{|E|}$ term matches.
   - Hence S_V is a vertex cover.

Some very similar NP-complete problems

Given a graph $G = (V, E)$

Definition (Max independent set)

Does there exist a set $S \subseteq V$ of size $\geq k$ such that, for all $(u, v) \in E$, at most 1 of $u \in S$ and $v \in S$?

Definition (Max clique)

Does there exist a set $S \subseteq V$ of size $\geq k$ such that, for all $(u, v) \in S$, $(u, v) \in E$?

Definition (Min vertex cover)

Does there exist a set $S \subseteq V$ of size $\leq k$ such that, for all $(u, v) \in E$, at least 1 of $u \in S$ and $v \in S$?

From here: preview of next class

- P: Polynomial time
- NP: Nondeterministic polynomial time
- BPP: Probabilistic polynomial time, failure probability at most $1/3$.
  - PP: failure probability $< 1/2$.
- BQP: Probabilistic quantum polynomial time, failure probability at most $1/3$.
- PSPACE: Polynomial space
- EXPTIME: Exponential time

Relations of complexity classes

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$$

Relations of complexity classes

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$$

- Know: $P \neq EXPTIME$, $PSPACE \neq EXPSPACE$.

Relations of complexity classes

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$$

- Know: $P \neq EXPTIME$, $PSPACE \neq EXPSPACE$.
- That's about it.

Relations of complexity classes

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$$

- Know: $P \neq EXPTIME$, $PSPACE \neq EXPSPACE$.
- That's about it.

$$P \subseteq BPP \subseteq BQP \subseteq PSPACE$$

Relations of complexity classes

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \ldots$$

- Know: $P \neq EXPTIME$, $PSPACE \neq EXPSPACE$.
- That's about it.

$$P \subseteq BPP \subseteq BQP \subseteq PSPACE$$

- Most people expect: $P = BPP$, everything else $\subsetneq$.

Relations of complexity classes

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots$$

- Know: $P \neq EXPTIME$, $PSPACE \neq EXPSPACE$.
- That's about it.

$$P \subseteq BPP \subseteq BQP \subseteq PSPACE$$

- Most people expect: $P = BPP$, everything else $\subsetneq$.
- Don't know NP compared to BPP or BQP (or even if one is inside the other).

Prototypical examples

- P: evaluate a *function*
  - Given a circuit f and input x, what is $f(x)$?
- NP: solve a *puzzle*
  - SAT: given f, determine if $\exists x : f(x) = 1$?
  - Given a puzzle, find the solution
  - Easy to verify once the solution is found.
- PSPACE: solve a *2-player game*
  - TQBF: $\exists x_1 \forall x_2 \exists x_3 \cdots \forall x_n : f(x) = 1$
  - Think chess: do I have a move, so no matter what you do, I can find a move, so no matter, etc., etc., I end up winning?
- Caveat: requires the puzzle/game to only have a *polynomial number of moves*.
  - Puzzles/games with exponentially many moves may be harder.
  - Go: actually EXPTIME-complete to solve a position.
  - Zelda: actually PSPACE-complete to solve a level.