Objectives

- Reducibility
- Conclusions

Classify Problems According to Computational Requirements

Fundamental Question:
Which problems will we be able to solve in practice?

Classify Problems

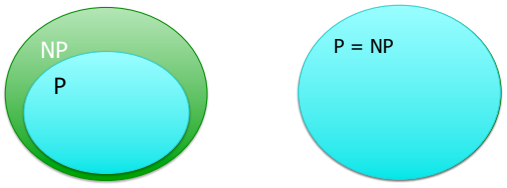
Classify problems according to those that can be solved in polynomial-time and those that cannot.

Frustrating news:
Many problems have defied classification.

Chapter 8. Show that problems are "computationally equivalent" and appear to be manifestations of one *really hard* problem.

Examples:
- Given a Turing machine, does it halt in at most *k* steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

The Big Question



P ⊆ NP

P = NP

In the mean time…

Classify problems according to those that can be solved in polynomial-time and those that cannot.

Frustrating news:
Many problems have defied classification.

Chapter 8. Show that problems are "computationally equivalent" and appear to be manifestations of one *really hard* problem.

Examples:
- Given a Turing machine, does it halt in at most *k* steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

Polynomial-Time Reduction

Suppose we could solve Y in polynomial time.
What else could we solve in polynomial time?

Polynomial-Time Reduction

Suppose we could solve Y in polynomial-time. What else could we solve in polynomial time?

- Reduction. Problem *X polynomial reduces to* problem *Y* if arbitrary instances of problem *X* can be solved using:
  - Polynomial number of standard computational steps, ***plus***
  - Polynomial number of calls to **oracle** that solves problem Y
    - Assume have a black box that can solve Y

For X + Y

- Notation: $X \leq_P Y$
  - "X is polynomial-time reducible to Y"
- Conclusion: If Y can be solved in polynomial time and $X \leq_P Y$, then X can be solved in polynomial time.

Fun Fact: Connecting Chapters 7 and 8

- Karp, of the Edmonds-Karp algorithm (max-flow problem on networks), published a paper in complexity theory on "Reducibility Among Combinatorial Problems", in which he proved 21 Problems to be NP-complete

Review: Polynomial-Time Reduction

Suppose we could solve Y in polynomial-time. What else could we solve in polynomial time?

- Reduction. Problem *X polynomial reduces to* problem *Y* if arbitrary instances of problem *X* can be solved using:
  - Polynomial number of standard computational steps, ***plus***
  - Polynomial number of calls to **oracle** that solves problem Y
    - Assume have a black box that can solve Y

For X + Y

- Notation: $X \leq_P Y$
  - "X is polynomial-time reducible to Y"
- Conclusion: If Y can be solved in polynomial time and $X \leq_P Y$, then X can be solved in polynomial time.

NP-Complete Problems

- Problems from many different domains whose complexity is unknown
- NP-completeness and proof that all problems are equivalent is **POWERFUL**!
  - All open complexity questions ➔ **ONE** open question!
- What does this mean?
  - "Computationally hard for practical purposes, but we can't prove it"
  - If you find an NP-Complete problem, you can stop looking for an efficient solution
    - Or figure out efficient solution for ALL NP-complete problems

Polynomial-Time Reduction

- Purpose. Classify problems according to *relative difficulty*.
- Design algorithms. If $X \leq_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.
- Establish intractability. If $X \leq_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.
- Establish equivalence. If $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$.

Considering $X \leq_P Y$

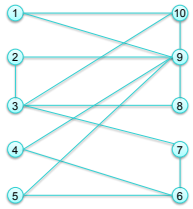
- Need to be careful putting X in terms of Y
- Make sure you're not putting an easy problem (X) in terms of a hard problem (Y)
  - While you could do that, what does that do for you?
  - Just because Y is hard to solve does *not* mean that X is hard to solve

Basic Reduction Strategies

- *Reduction by simple equivalence*
- Reduction from special case to general case
- Reduction by encoding with gadgets

Independent Set

- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices S ⊆ V such that |S| ≥ *k* and for each edge **at most one** of its endpoints is in *S*?

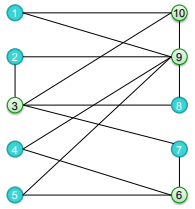


How is this different from the network flow problem?

Ex. Is there an independent set of size ≥ 6?
Ex. Is there an independent set of size ≥ 7?

Independent Set

- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices S ⊆ V such that |S| ≥ *k* and for each edge **at most one** of its endpoints is in *S*?

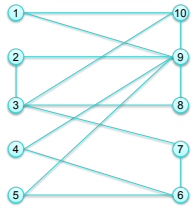


Ex. Is there an independent set of size ≥ 6? Yes
Ex. Is there an independent set of size ≥ 7? No

independent set

Vertex Cover

- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices S ⊆ V such that |S| ≤ *k* and for each edge, **at least one** of its endpoints is in S?



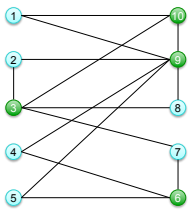
A vertex ***covers*** an edge.

Application: place guards within an art gallery so that all corridors are visible at any time

Ex. Is there a vertex cover of size ≤ 4?
Ex. Is there a vertex cover of size ≤ 3?

Vertex Cover

- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices S ⊆ V such that |S| ≤ *k* and for each edge, **at least one** of its endpoints is in S?



Ex. Is there a vertex cover of size ≤ 4? Yes
Ex. Is there a vertex cover of size ≤ 3? No

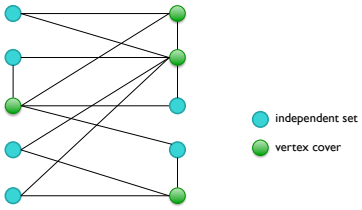
vertex cover

Problem

- Not known if finding Independent Set or Vertex Cover can be solved in polynomial time
- BUT, what can we say about their relative difficulty?

Vertex Cover and Independent Set

- Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET
- Pf. We show *S* is an independent set iff V - S is a vertex cover



Vertex Cover and Independent Set

- Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET
- Pf. We show S is an independent set iff V - S is a vertex cover
- $\Rightarrow$
  - Let S be an independent set
  - Consider an arbitrary edge (u, v)
  - Since S is an independent set $\Rightarrow$ $u \notin S$ or $v \notin S$ or both $\notin S$ $\Rightarrow$ $u \in V - S$ or $v \in V - S$ or both $\in V - S$
  - Thus, V - S covers (u, v)
    - Every edge has at least one end in V-S
  - V-S is a vertex cover

Vertex Cover and Independent Set

- Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET
- Pf. We show S is an independent set iff V - S is a vertex cover
- $\Leftarrow$
  - Let V - S be any vertex cover
  - Consider two nodes $u \in S$ and $v \in S$
  - Observe that $(u, v) \notin E$ since V - S is a vertex cover
  - Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set

Using the Previous Result

- Problem *X* *polynomial reduces to* problem *Y* if arbitrary instances of problem *X* can be solved using:
  - Polynomial number of standard computational steps, ***plus***
  - Polynomial number of calls to **oracle** that solves problem Y
    - Assume have a black box that can solve Y

How do we show polynomial reduction for the independent set and vertex cover?

Summary

- If we have a block box to solve Vertex Cover, can decide whether G has an independent set of size at least *k* by asking the black box whether G has a vertex cover of size at most $n - k$
- If we have a block box to solve Independent Set, can decide whether G has a vertex cover of size at most *k* by asking the block box whether G has an independent set of size at least $n - k$

Basic Reduction Strategies

- Reduction by simple equivalence
- *Reduction from special case to general case*
- Reduction by encoding with gadgets

Set Cover

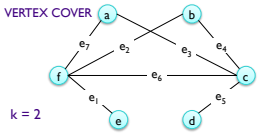
- SET COVER: Given a set U of elements, a collection S_1, S_2, . . . , S_m of subsets of U, and an integer k, does there exist a collection of size ≤ k of these sets whose union is equal to U?
- Sample application
  - m available pieces of software
  - Set U of n capabilities that we would like our system to have
  - The i^{th} piece of software provides the set $S_i \subseteq U$ of capabilities
  - **Goal**: achieve all n capabilities using fewest pieces of software
- Ex:

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2
S_1 = {3, 7}
S_2 = {3, 4, 5, 6}
S_3 = {1}
S_4 = {2, 4}
S_5 = {5}
S_6 = {1, 2, 6, 7}

Choose S_2 and S_6

Vertex Cover Reduces to Set Cover

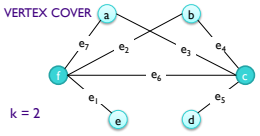
- Claim. VERTEX-COVER ≤P SET-COVER
- Pf. Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.
  - …



SET COVER?

Vertex Cover Reduces to Set Cover

- Claim. VERTEX-COVER ≤P SET-COVER
- Pf. Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.
  - …

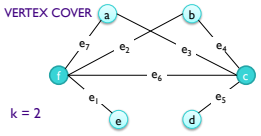


SET COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2
S_a = {3, 7}
S_b = {2, 4}
S_c = {3, 4, 5, 6}
S_d = {5}
S_e = {1}
S_f = {1, 2, 6, 7}

Vertex Cover Reduces to Set Cover

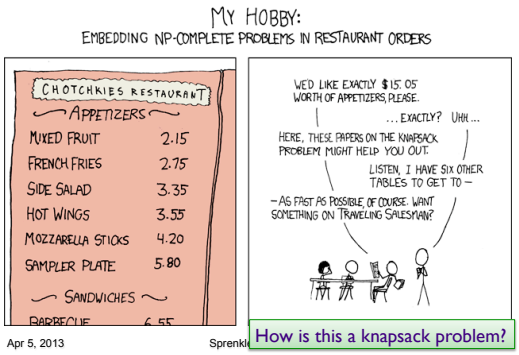
- Claim. VERTEX-COVER ≤ P SET-COVER
- Pf. Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.
- Construction.
  - Create SET-COVER instance:
    - k = k, U = E, S_v = {e ∈ E : e incident to v }
  - Set-cover of size ≤ k iff vertex cover of size ≤ k. ▪



SET COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2
S_a = {3, 7}
S_b = {2, 4}
S_c = {3, 4, 5, 6}
S_d = {5}
S_e = {1}
S_f = {1, 2, 6, 7}

Now you "get" this xkcd comic



How is this a knapsack problem?

PS8

- Dynamic programming
- Follow examples from class/book
  - Recurrence relation/subproblems
    - Explain intuition
  - Use memoization
  - Process to find *solution* after finding *value*
  - Analysis of runtime

Final

- Usual rules
- Due next Friday, 5 p.m. (end of exams)
- Can use book, notes, handouts, my lecture notes, me (limited)
  - "The status of the P versus NP problem"
  - No other outside resources
- Office hours: Tuesday, 9:10 a.m.-11:50 a.m., 1:30 p.m. – 3:20 p.m.
  - Others by appointment
- Evaluations due Monday at midnight on Sakai (tests and quizzes)
  - Last checked: 4 submissions of evaluations