Objectives

- Oh, the places you've been!
- Oh, the places you'll go!

> Now, everything comes down to expert knowledge of **algorithms** and **data structures**. If you don't speak fluent **O-notation**, you may have trouble getting your next job at the technology companies in the forefront.
> — Larry Freeman

Algorithm Design Patterns

- What are some approaches to solving problems?
- How do they compare in terms of difficulty?

Algorithm Design Patterns

- Greedy
- Divide-and-conquer
- Dynamic programming
- Duality/network flow

Course Objectives: Given a problem…

You'll recognize when to try an approach
- AND, when to bail out and try something different

Know the steps to solve the problem using the approach
- e.g., breaking it into subproblems, sorting possibilities in some order

Know how to ***analyze*** the run time of the solution
- e.g., solving recurrence relation

Algorithm Design Patterns

- Greedy
- Divide-and-conquer
- Dynamic programming
- Duality/network flow
- Reductions – Chapter 8
- Local search – Chapter 12
- Randomization – Chapter 13

What Was Our Goal In Finding a Solution?

Polynomial Time → Efficient

POLYNOMIAL-TIME REDUCTIONS

Classify Problems According to Computational Requirements

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

Classify Problems According to Computational Requirements

Which problems will we be able to solve in practice?

- Working definition. [Cobham 1964, Edmonds 1965, Rabin 1966] Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

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?

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.

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

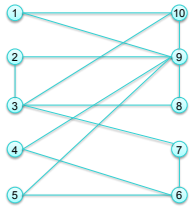
- Purpose. Classify problems according to *relative difficulty*.
- Design algorithms. If $X \leq_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.
- Establish intractability. If $X \leq_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in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

Basic Reduction Strategies

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

Independent Set

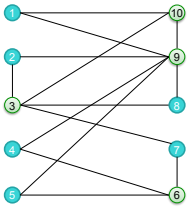
- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$ and for each edge **at most one** of its endpoints is in *S*?



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 $S \subseteq V$ such that $|S| \geq k$ and for each edge **at most one** of its endpoints is in *S*?

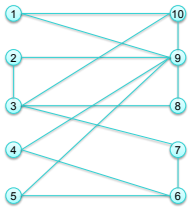


Ex. Is there an independent set of size ≥ 6? Yes
Ex. Is there an independent set of size ≥ 7? No

independent set

Vertex Cover

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



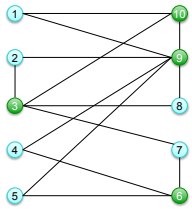
A vertex ***covers*** an edge.

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 $S \subseteq V$ such that $|S| \leq k$ 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

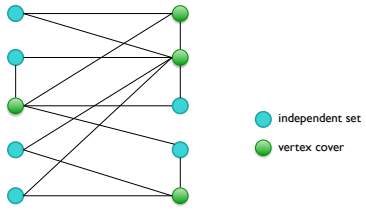
vertex cover

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, then we 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, then we 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

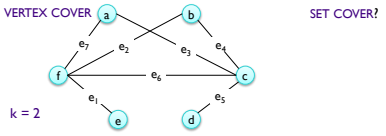
- SET COVER: Given a set U of elements, a collection S_1, S_2, . . . , S_m of subsets of U, and an integer k, does there exist a collection of size ≤ k of these sets whose union is equal to U?
- Sample application
  - m available pieces of software
  - Set U of n capabilities that we would like our system 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

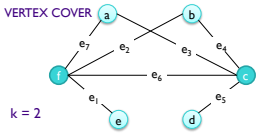
Vertex Cover Reduces to Set Cover

- 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.
  - …



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.
  - …

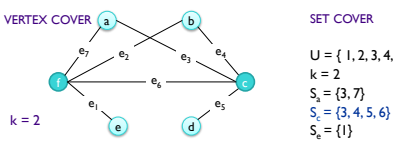


SET COVER

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}

Overview of Showing NP Completeness

1. Show that X is in NP.
   - Argue that there is an efficient certifier for X (Brief but necessary step)
     - i.e., there is a certifier s.t. for any yes instance of X, there exists a certificate that the certifier will accept, and for any no instance of X, there is no certificate that the certifier will accept
     - The running time of the certifier (and hence the size of the certificate) must be polynomial.
     - Typically, a solution to the given problem is a sufficient certificate
2. Pick a known NP-complete problem. State what problem Y you are reducing to X.
   - Need to show that Y ≤$_P$ X.
   - Use any problem Y proved to be NP-complete
   - By defn, if X is NP-complete, then you can use any other NP-complete problem Y to show this. Some problems will be easier to use than others. Decide which NP-complete problem is most natural.
3. Construct an algorithm to solve Y given an algorithm to solve X.
   - Show that any instance of Y can be solved using a polynomial number of operations, and a polynomial number of calls to a black box that can solve X.
   - It is very easy to get mixed up and instead prove that X ≤p Y . Typically: show how to solve Y by constructing a single input to the black box for X, and your algorithm will output the same answer as is given by the black box, but this is not always the case.
4. Prove the correctness of your algorithm.
   - Prove an *iff* statement. Show that given a yes instance of Y algorithm returns "yes", and show that if your algorithm returns "yes," then the given input is indeed a yes instance of Y .
   - Trivial to come up with an algorithm that satisfies just one of these two conditions. Want something that satisfies both.
5. Polytime and wrap-up.
   - Conclude that since algorithm runs in polynomial time, Y ≤$_P$ X.
   - Since Y is NP-complete, and since we also have shown that X is in NP, X is NP-complete.

Planning

- For Friday
  - Problem set
  - Post on Sakai (5 pts)
    - Brief overview statement (what is the article about)
    - 3 most important points
    - Questions: either for discussion or for understanding
  - EC Airline Scheduling problem: 3 pts
- Total problem set points for semester: 222
  - Fill out course evaluations on Sakai
  - If 60% fill out, 1% EC on problem sets
    - Additional 1% for every additional 12.5% who complete
  - Due Sunday at midnight