Objectives

- Algorithms Retrospective
- Computational intractability

Review

- What is the power of the max-flow/min-cut algorithm?
- What is our process in solving problems using network flow?

Review: Network Flow Solutions

1. Model problem as a flow network
   - Describe what nodes, edges, and capacity represent
   - Describe what flow represents and how that maps to your solution
   - Run Ford-Fulkerson algorithm
     - Map back to original problem
2. Prove that the solution found is correct/feasible/optimal
3. Prove that you find all solutions
4. Analyze running time
   - Creating model
   - FF algorithm

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

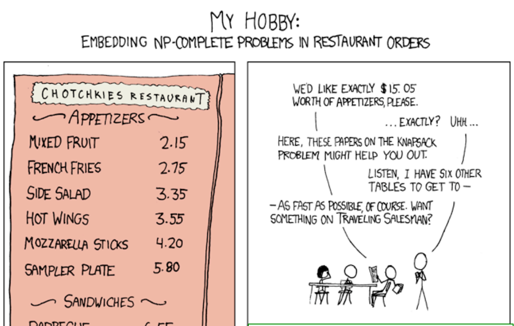
My Algorithms Approach

- Why problems?
- Why wiki?
- Research to support decisions

Algorithm Design Patterns

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

Now you "get" this xkcd comic



How is this a knapsack problem?

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.

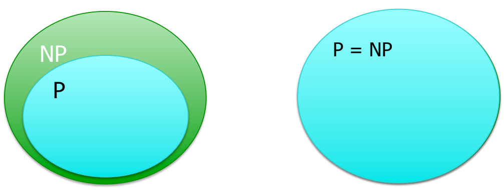
Frustrating news: *Many* problems have defied classification.

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?

The Big Question

NP: "nondeterministic polynomial time"



$P \subseteq NP$

$P = NP$

Are there polynomial-time solutions to NP problems?

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"
    - proved 21 Problems to be NP-complete

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 $X \leq_P Y$

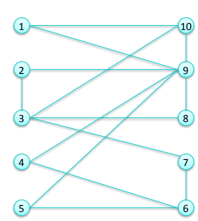
- Need to be careful putting X in terms of Y
- Make sure you're not putting an easy problem (X) in terms of a hard problem (Y)
  - While you could do that, what does that do for you?
  - Just because Y is hard to solve does *not* mean that X is hard to solve

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

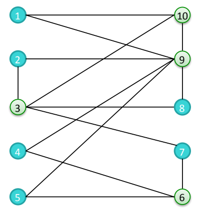


How is this different from the network flow problem?

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*?

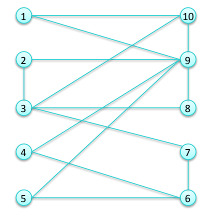


Ex. Is there an independent set of size ≥ 6? Yes
Ex. Is there an independent set of size ≥ 7? No

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?



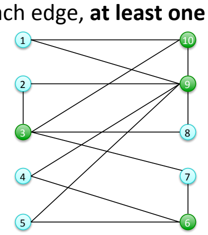
A vertex ***covers*** an edge.

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

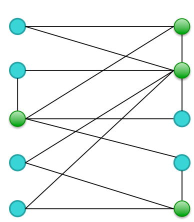
vertex cover

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



independent set
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 ∉ S or v ∉ S or both ∉ S $\Rightarrow$ u ∈ V - S or v ∈ V - S or both ∈ 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*

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", Chicago Mag article
  - No other outside resources
- Office hours:
  - Monday: 10 a.m.– 5 p.m.
  - Tuesday: 9:10 a.m. – 5 p.m.
  - Thursday: 9:10 a.m. – 2:30 p.m.
  - Appointments preferable during that time
  - Others by appointment
  - Can email about other appointments as necessary
- Evaluations due Sunday at midnight on Sakai (tests and quizzes)