Midterm

Time: Apr. 15 Thursday, in class (10:15-11:55am)
Location: 教学中心 201 & 301.
Seating arrangement TBA.

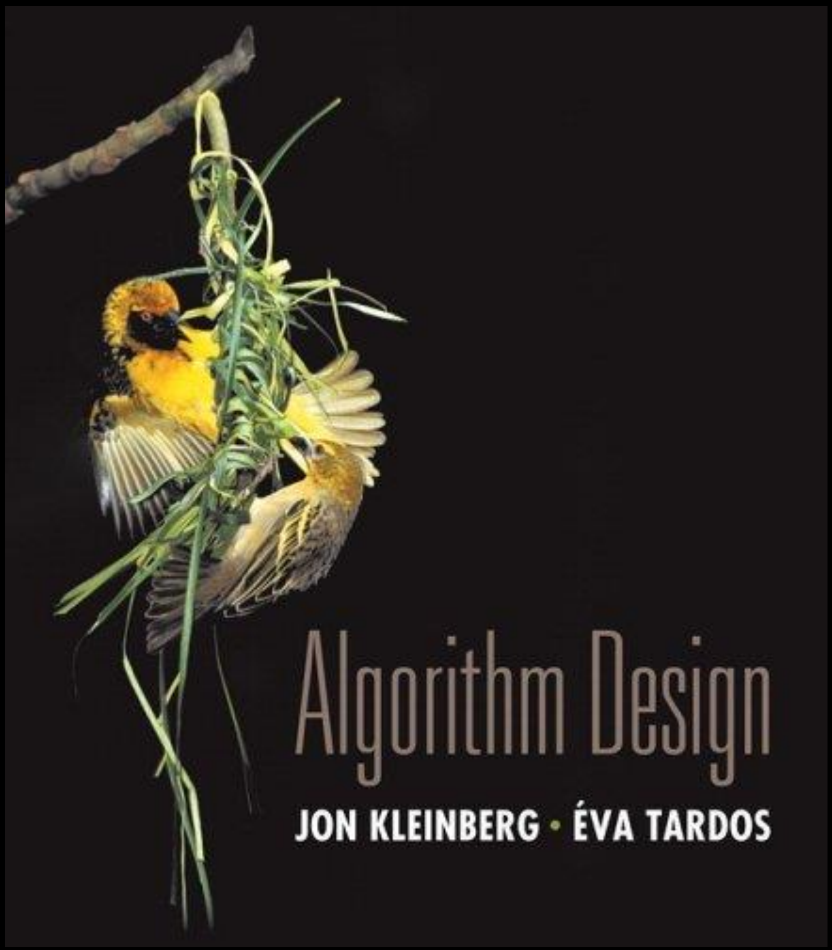
Covers Chapter 2, 4~8

Format:

- 5 multi-choices + 4 problems;
- closed-book, one A4-size cheat sheet allowed

Grade: %35 of the total grade

Midterm Review



Chapter 2

Basics of Algorithm Analysis

Slides by Kevin Wayne.

Basics of Algorithm Analysis

Worst case analysis. Obtain bound on largest possible running time of algorithm on input of a given size N.

Asymptotic Order of Growth

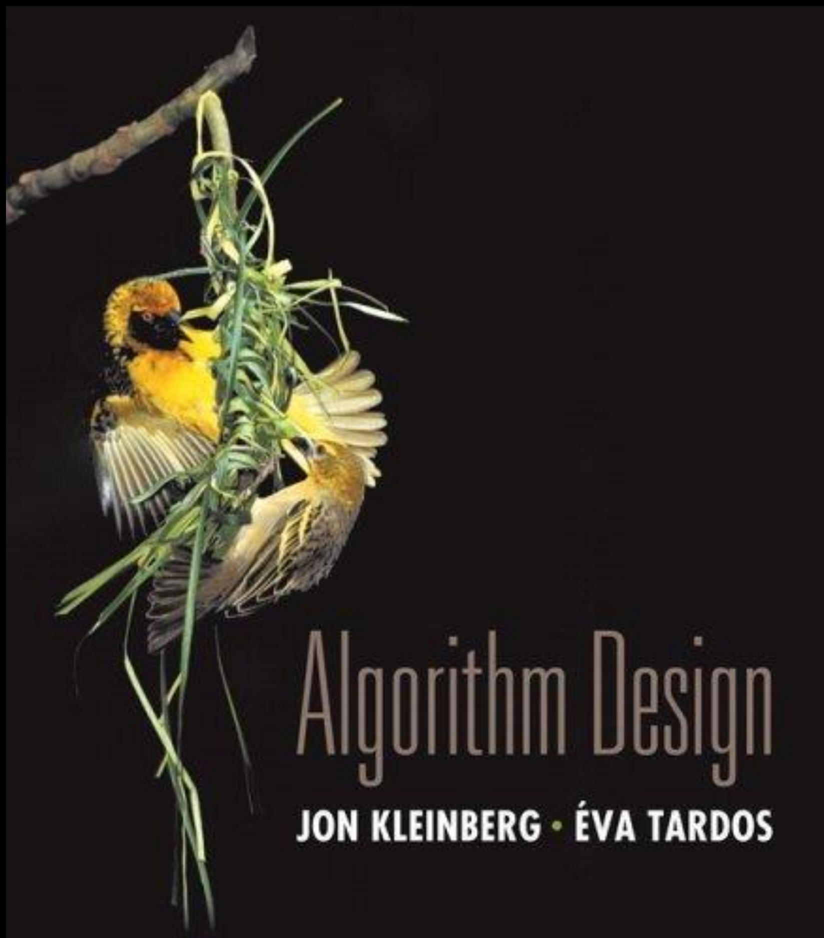
- Upper bounds. $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c \cdot f(n)$.
- Lower bounds. $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \geq c \cdot f(n)$.
- Tight bounds. $T(n)$ is $\Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

Asymptotic Bounds for Some Common Functions

Polynomials. $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.

Logarithms. For every $x > 0$, $\log n = O(n^x)$.

Exponentials. For every $r > 1$ and every $d > 0$, $n^d = O(r^n)$.



Chapter 4

Greedy Algorithms

Slides by Kevin Wayne.

Greedy Algorithms

Basic idea

- Make the locally optimal choice at each step.

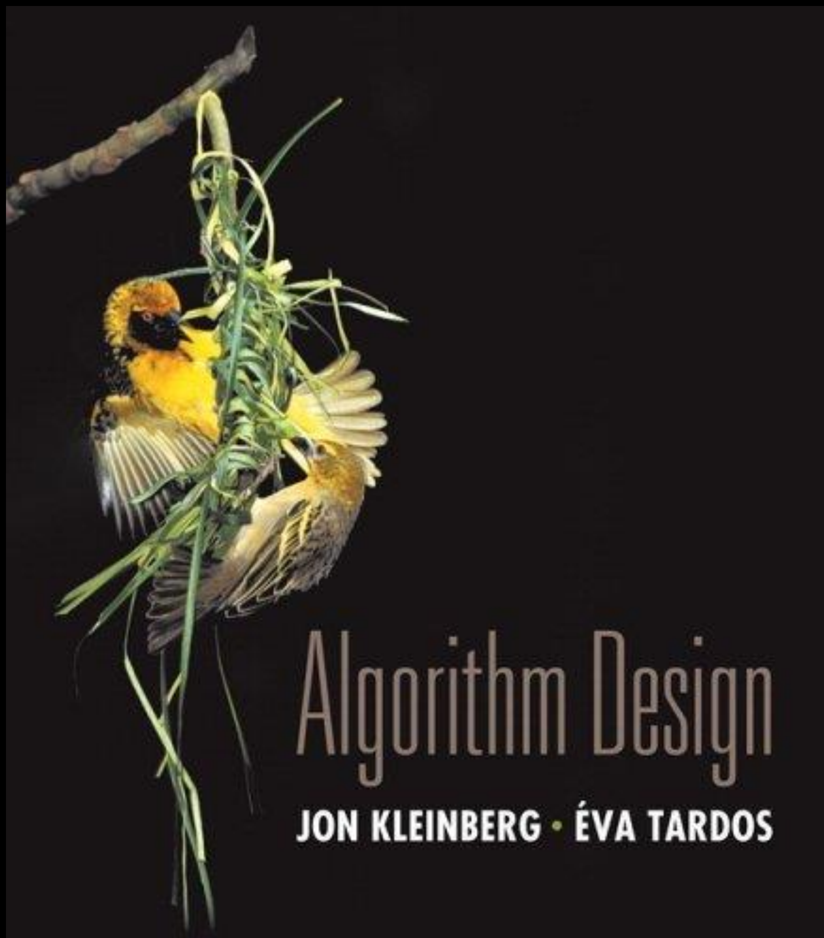
Algorithms

- Interval Scheduling
  - Choose the job with the earliest finish time
- Scheduling to Minimize Lateness
  - Choose the job with the earliest deadline
- Optimal Caching
  - Evict item that is requested farthest in future
- Clustering
  - Single-link k-clustering

Greedy Algorithms

Proof skills

- Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.
- Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.



Chapter 5

Divide and Conquer

Slides by Kevin Wayne.

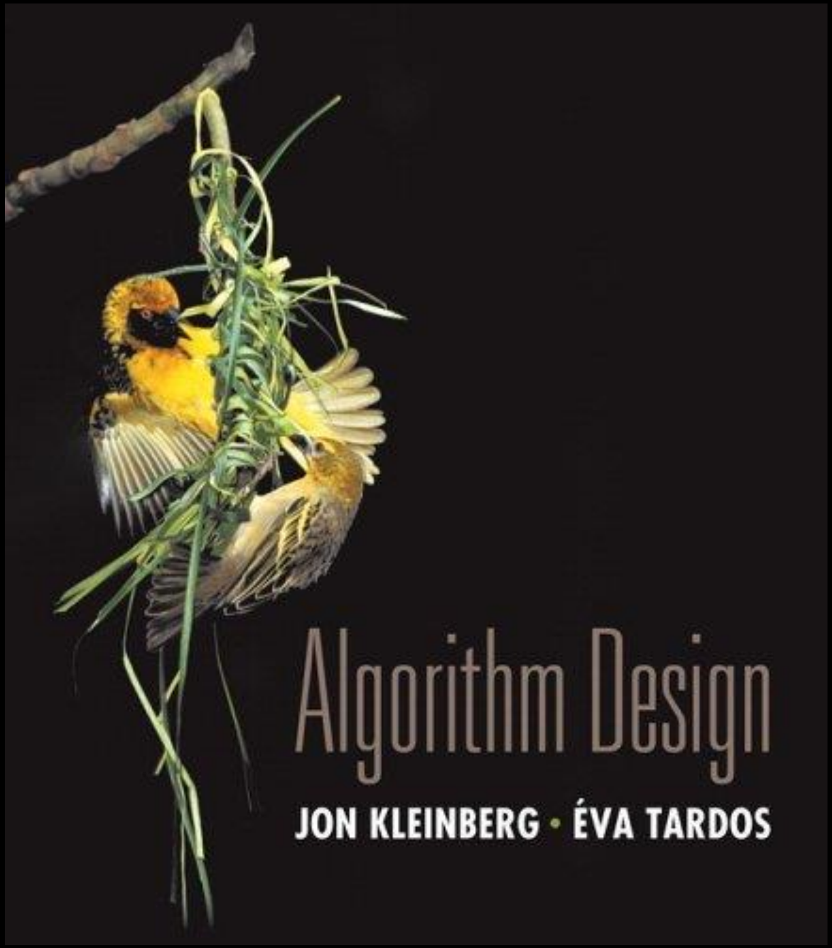
Divide-and-Conquer

Basic idea

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Algorithms

- Mergesort
  - Divide a sequence into two of same size
- Closest Pair of Points
  - Vertically divide the space
- Integer Multiplication
  - Divide each n-digit integer into two $\frac{1}{2}$n-digit integers
- Matrix Multiplication
  - Divide each n-by-n matrix into four $\frac{1}{2}$n-by-$\frac{1}{2}$n blocks
- Fast Fourier Transform
  - Divide a polynomial into two with even and odd powers



Chapter 6

Dynamic Programming

Slides by Kevin Wayne.

Dynamic Programming

Basic idea

- Polynomial number of sub-problems with a natural ordering from smallest to largest.
- Optimal solution to a sub-problem can be constructed from optimal solutions of smaller sub-problems.
- Sub-problems are overlapping!

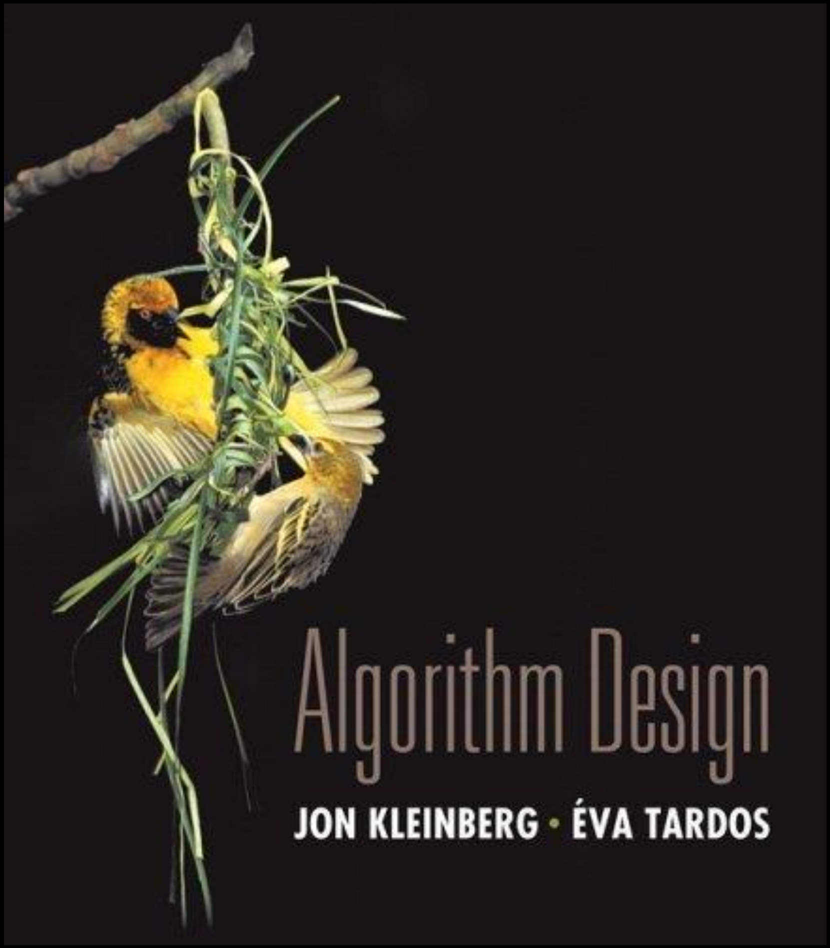
Guideline

- Define the sub-problems
  - OPT(…)
- Write down the recursive formulas
  - Ex: $OPT(i) = \max(f(OPT(j)), g(OPT(k)), \ldots),\ j,k < i$
- Compute the formulas either bottom-up or top-down

Dynamic Programming

Algorithms

- Weighted interval scheduling
  - 1D array; binary choice
- Knapsack
  - 2D array; adding a new variable (weight limit)
- RNA secondary structure
  - 2D array: intervals
- Sequence Alignment
  - 2D array: prefix alignment
- Sequence Alignment in Linear Space
  - Combination of divide-and-conquer and dynamic programming
- Shortest path with negative edges
  - (Bellman-Ford) 2D array: shortest path with edge number $\leq i$
- Distance Vector Protocol
- Negative Cycle Detection



Chapter 7

Network Flow

Slides by Kevin Wayne.

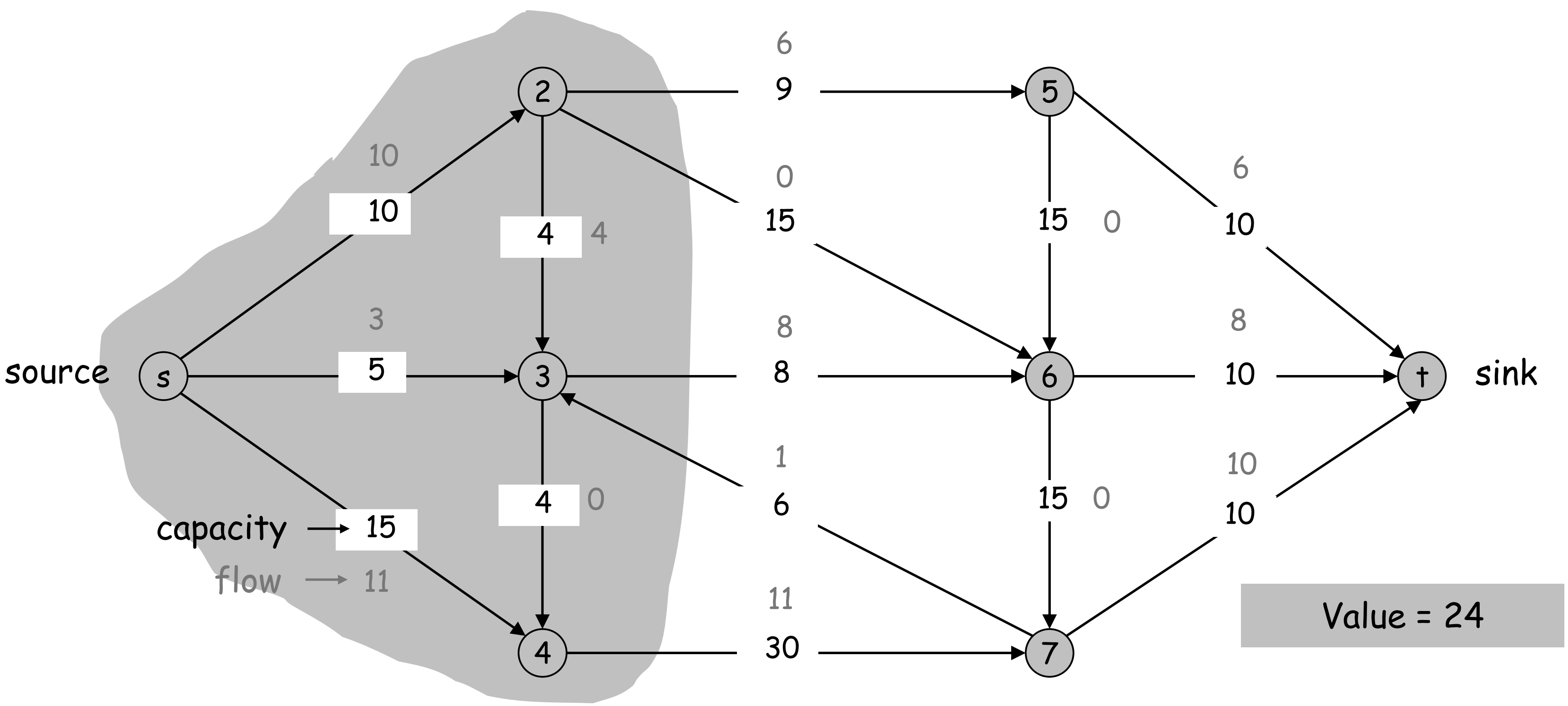
Flows

Concepts

- s-t flow
- Max-flow
- s-t cut
- Min-cut

Max-flow min-cut theorem.

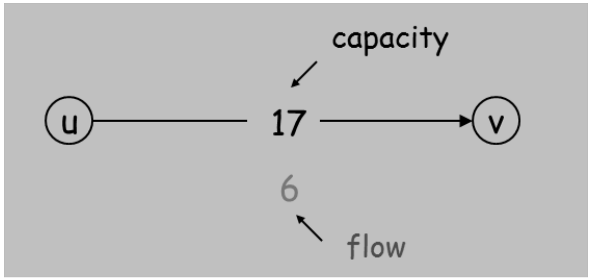
The value of the max flow is equal to the value of the min cut.

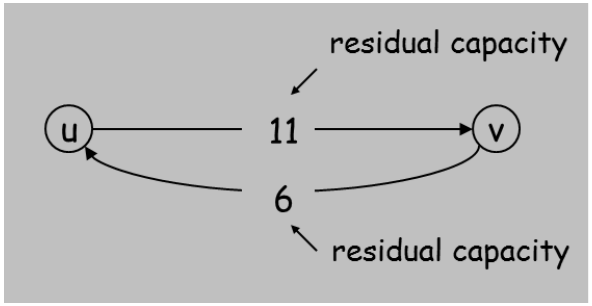


Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an augmenting path P in the residual graph G_f.
  - Can be chosen using capacity scaling
- Augment flow along path P.
- Repeat until you get stuck.





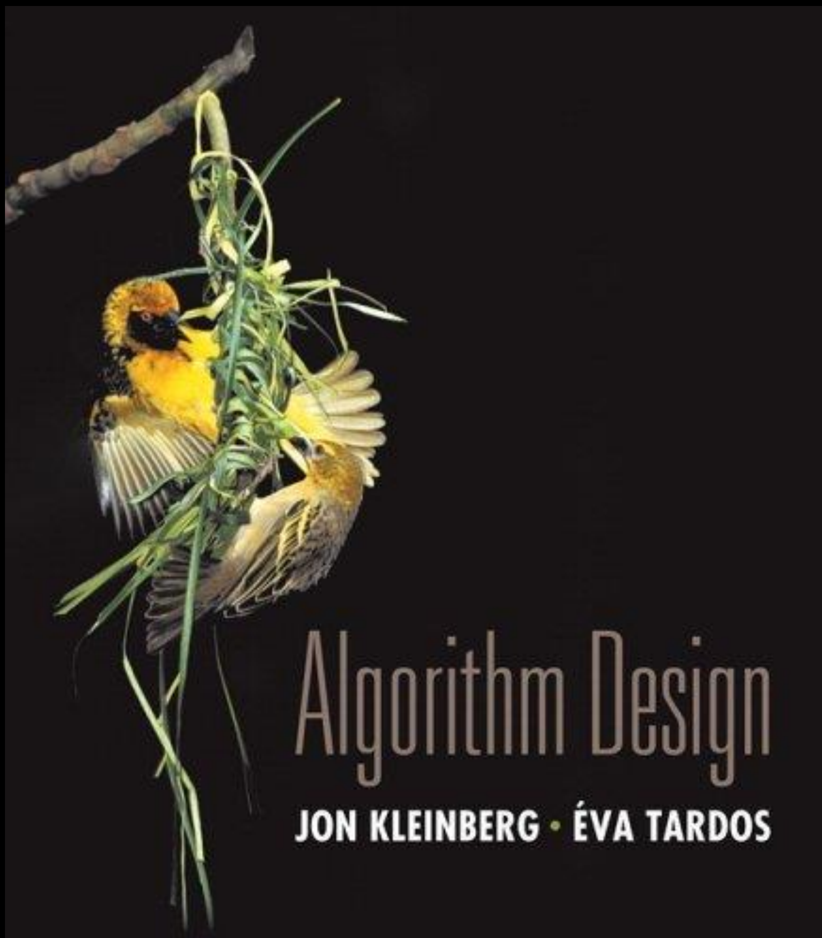

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
      update Gf
   }
   return f
}
```

Applications

Problems covered in class

- Bipartite Matching
- Circulation with Demands (+ edge lower bounds)
- Survey Design
- Image Segmentation
- Project Selection
- Baseball Elimination



Chapter 8

NP and Computational Intractability

Slides by Kevin Wayne.

Key Concepts

Decision problem

- Answer yes/no

P. Decision problems for which there is a poly-time algorithm.

NP. Decision problems for which there exists a poly-time certifier.

- Algorithm $C(s, t)$ is a certifier for problem X if for every string s, $s \in X$ iff there exists a string t such that $C(s, t) =$ `yes`.

co-NP. Complements of decision problems in NP.

EXP. Decision problems for which there is an exponential-time algorithm.

Claim. $P \subseteq NP$, $\text{co-NP} \subseteq EXP$

Polynomial-Time Reduction

Reduction. Problem X polynomial-time 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.

Notation. $X \leq_P Y$.

Common approach: polynomial transformation

- Given any input x to X, construct an input y in poly-time such that x is a `yes` instance of X iff y is a `yes` instance of Y.

NP-Completeness

NP-complete. A problem Y in NP with the property that for every problem X in NP, $X \leq_p Y$.

Recipe to establish NP-completeness of problem Y.

- Step 1. Show that Y is in NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that $X \leq_p Y$.

NP-Completeness