• Flows and Matchings
  • Max flow, Ford Fulkerson
  • Max Flow Min Cut theorem
  • Edmonds-Karp, Dinic’s algorithm
  • Bipartite Matching, Hall’s theorem
  • Min Cost Bipartite Matching
• Linear Programming
  • LP duality
  • Complementary slackness, strong duality
  • Minimax theorem
  • Algorithms for LP
Online Algorithms

- Multiplicative Weights algorithms
- Applications of multiplicative weights to Games and LP
- Online bipartite matching
- Random arrivals
- K-server
• **Approximation Algorithms**
  • Max coverage, Set Cover, TSP
  • Randomized rounding, Edge disjoint paths
  • Semidefinite programming for Max Cut
  • LP based algorithms for Set Cover
  • Primal-dual algorithm for Facility Location
  • K-means++
• Glimpses of topics beyond 261
  • Beyond worst case analysis: stability
    • Smoothed analysis: interpolating between worst case and average case analysis

  • Distributed algorithms
    • Massively Parallel Computation (MPC), MapReduce

  • Fine grained complexity
    • Can we improve on $O(n^2)$ dynamic programming algorithm for edit distance?

• Today: Sketching and Streaming algorithms
How many bits do you need to count from 1 to n?

- Counter takes log n bits. Can we do better?
- **Desiderata:** After k increment operations, output a value between k(1 - \(\varepsilon\)) and k(1 + \(\varepsilon\)) with probability 1 - \(\delta\)
- \(O(\log \log n)\) bits suffice!
- Maintain “log” counter \(X\)
- Increment with probability \(2^{-X}\)
- \(E[2^X] = k+1\), where \(k = \) number of increments
Sketching and Streaming Algorithms

• Algorithmic toolkit for very large data sets
  • How to design algorithms for data sets that are so large that they do not fit into memory?
  • **Streaming algorithms**: Make one pass (or multiple passes) over data to perform computation
  • Linear or near linear time algorithms
  • **Sketching**: replace each data item by a much smaller sketch; operate on sketches instead
  • **Approximation**: Compute approximations instead of exact quantities of interest
  • **Randomization**: Algorithms work with high probability
Counting Distinct Elements

• Estimate the number of distinct queries in Google’s query log
  • How would you do it?
  • What if the available memory is not large enough to store hash table with all queries?
Counting Distinct Elements

• Clever use of hashing!
• Scale hash values so they are in the range [0,1]
• Assume $h(x)$ is uniform random number in [0,1]
• Also assume hash values are independent
• Key idea: $Y = \min (h(x_1), h(x_2), ... h(x_k))$
• What is $E[Y]$?
• \( \Pr[Y > t] = (1-t)^k \)

• \( E[Y] = \int_0^1 \Pr[Y > t] \, dt \)

• \( = \int_0^1 (1 - t)^k \, dt \)

• \( = \frac{1}{k+1} \)

• What about \( \text{Var}[Y] \) ?

• \( \text{Var}[Y] \leq \frac{1}{(k+1)^2} = E[Y]^2 \)

• Chebyshev?

• \( \Pr[|Y - E[Y]| > \varepsilon \cdot E[Y]] \leq \frac{\text{Var}[Y]}{(\varepsilon \cdot E[Y])^2} \leq \frac{1}{\varepsilon^2} \)

• Not useful!
Counting Distinct Elements

• Track multiple independent versions $Y_1 \ldots Y_t$ of $Y$

• Use mean $Z = \frac{Y_1 + Y_2 + \cdots + Y_t}{t}$

• $\text{E}[Z] = \text{E}[Y]$  

• $\text{Var}[Z] = \frac{1}{t^2} \sum_{1}^{t} \text{Var}[Y_i] = \frac{\text{Var}[Y]}{t} \leq \frac{(\text{E}[Y])^2}{t}$

• $\text{Var}[Z] \leq \frac{(\text{E}[Z])^2}{t}$

• $\text{Pr}[[Z - \text{E}[Z]] > \varepsilon \cdot \text{E}[Z]] \leq \frac{\text{Var}[Z]}{(\varepsilon \cdot \text{E}[Z])^2} \leq \frac{1}{\varepsilon^2 t}$

• $t = \frac{4}{\varepsilon^2}$ Failure probability $= \frac{1}{4}$

• $t = \frac{1}{\varepsilon \delta}$ Failure probability $= \delta$
Can we reduce the error probability?

Multiple independent versions \( Z_1 \ldots Z_s \) of \( Z \)

Return median(\( Z_1 \ldots Z_s \))

Each \( Z_i \) is bad with probability at most \( 1/4 \)

How big does \( s \) need to be for median to be good?

Median is bad if at least \( s/2 \) \( Z_i \) are bad

Expected number of bad \( Z_i \) is at most \( s/4 \)

Chernoff bounds!

\[
\Pr[\text{median is bad}] \leq e^{-s/11}
\]

Suffices to set \( s = 11 \ln\left(\frac{1}{\delta}\right) \)
Counting Distinct Elements

• Fix random hash function assumption
• Use pairwise independent hash function
• \( h_{a,b}(x) = ax + b \pmod{p} \)
• Constant factor approximation to distinct elements
• To get \((1 + \varepsilon)\) approximation, store \( O\left(\frac{1}{\varepsilon^2}\right)\) smallest hash values
• **Practical implementation: HyperLogLog**
• Estimate = $2^{\text{number of trailing 1's in hash value}}$
• Split data into groups
• Geometric mean of estimate from each group
Sketching for similarity estimation

• Vector representation of documents
• Distance measured by angle between vectors
• Computing angle requires time proportional to size of the documents
• Construct sketch for vectors to estimate angle?
  • Simhash
  • Inspiration from Goemans-Williamson random hyperplane rounding
  • Pick random vectors \( r_1 \ldots r_t \)
  • For vector \( v \) store \( \text{sign}(v.r_i) \): t bit sketch
  • Hamming distance between sketches used to estimate angles
Theory Courses

- **CS 368 (Spring):** Algorithmic Techniques for Big Data
- **CS 265 (Fall):** Randomized Algorithms
- **CS 250 (Winter):** Error Correcting Codes
- **CS 260 (Winter):** Geometry of Polynomials in Algorithm Design
- **CS 259Q (Winter):** Quantum Computing
- **CS 354 (Spring):** Topics in Intractability
- **MS&E 315 (Spring):** Advanced Optimization Theory
- **MS&E 316 (Winter):** Discrete Mathematics and Algorithms
- **MS&E 319 (Fall, Spring):** Matching Theory