Overview

Last Class
Paradigms for Randomized Algorithms
  - Foiling an Adversary
  - Abundance of Witnesses
  - Sampling
  - Markov Chains

Today
  1) More Paradigms for Randomized Algorithms
  2) Complexity Classes for Randomized Algorithms

Paradigms for Randomized Algorithms

Fingerprinting/Hashing/Load Balancing

When testing the equivalency of two inputs of possibly complex size and shape, it is sometimes easier to randomly generate a function which translates the inputs to a simpler output. The simpler output can then be checked for equivalency.

In addition, minor variations in the inputs can create problems for deterministic equivalency algorithms. These variations can be naturally overlooked when the randomly generated function maps the inputs to the simpler output.

Problem Test is \( x = y \)
Solution Introduce randomly generated function, \( f \), check \( f(x) = f(y) \)

Symmetry Breaking

Randomness tends to generate evenly balanced distributions over time. In situations where there are many possible solution states, a randomized algorithm can be used to randomly select one of the possible solution states.

Problem Distributed Computations
  - Many Solutions
  - Need Algorithm to find one
Solution Use randomization to (implicitly) pick out a specific solution (eg. Perfect matchings)
  Give edges random weights (Unique lowest weight matching)
Random Reordering

Sometimes deterministic algorithms performance varies with the nature of the data. Randomly reordering the data can help to prevent data from causing a big slowdown.

**Problem**

Eg. Quicksort

**Solution**

Avoid bad inputs by random reordering

Random Constructions

When doing searches, randomly selecting data is a possible way to sift through a massive number of elements. The probability of selecting a bad set of data as compared to deterministic algorithms is very low.

**Problem**

Produce a prime number between n and 2n

**Solution**

Use Random Algorithm: number of primes \( \approx \frac{n}{\log n} \).

Randomly select number, check primality, and repeat.

\( O(\log n) \)

**Theorem**

There exists a graph on \( n \) vertices with no clique or independent set of size \( \geq 2 \log 2n \). Ramsey number, \( R(m, m) < 4^m \)

**Proof**

Take a random graph.

Complexity Classes (1.5)

Complexity classes formalize what it means for a randomized algorithm to be efficient. These classes can be used to show the complexity and reliability of our algorithms.

Vocabulary

Languages

Decision Problems

Complexity Classes

Set of Languages

General Requirements

Each algorithm must follow a certain set of criterion to be classified. The algorithm should:

1) Runs in polynomial time
2) Correct with > \( \frac{1}{2} \) probability

Probability Polynomial (PP)

*Probability Polynomial defines a complexity class where the algorithms must both accept their target language and reject other languages at least 50% of the time. This shows that the algorithm at least tends towards a solution, even if very slowly.*

**Definition**

Language \( L \in \text{PP} \)

If 3 polytime randomized algorithm, A, such that

- \( x \in L, \text{Pr}(A(x) \text{accepts}) > \frac{1}{2} \)
- \( x \notin L, \text{Pr}(A(x) \text{accepts}) < \frac{1}{2} \)
Theorem  \( NP \subseteq PP \)  **NOT VERY USEFUL**

Proof  Determine if \( F \) accepts any input - Pick random \( x \)

- if \( F(x) \) accepts then return YES
- else return YES with probability \( \frac{1}{2} - \epsilon \)

Note  This is not a very useful theorem because NP is already a large complexity class that includes almost all algorithms. To say that our complexity class is superset of NP is so broad that it does not help classify our algorithms. Being a part of PP does not give any indication about the algorithm.

Bounded Error Probability Polynomial (BPP)

_Bounded Error Probability Polynomial defines a complexity class where the algorithms must both accept their target language and reject other languages at least more than 50% of the time. This solves the problem with PP where the probability of accepting/rejecting is infinitesimally close to 50%. This causes slow, almost nonexistent, progress towards a solution. BPP will converge towards a solution faster depending on how far away from 50% the probability of accepting the target language and rejecting other languages is._

Definition  Language \( L \in BPP \)

If \( \exists \) polytime randomized algorithm, \( A \), such that

- \( x \in L, \Pr(A(x) \text{accepts}) > \frac{2}{3} \) or \( (1 - \epsilon) \)
- \( x \notin L, \Pr(A(x) \text{accepts}) < \frac{1}{3} \) or \( \epsilon \)

Run Algorithm \( \log \left( \frac{1}{\epsilon} \right) \) times with independent randomness and use most common value

Solovay-Strassen Test

- If Prime -> always passed
- If composite -> failed w/ > \( \frac{1}{2} \) probability.

Complexity Class: Randomized Polynomial Time (RP)