CMSC5706 Topics in Theoretical Computer Science

Week 1: Review of Algorithms and Probability

Instructor: Shengyu Zhang

First week

- Part I: About the course
- Part II: About algorithms and complexity
  - What are algorithms?
  - Growth of functions
  - What is the complexity of an algorithm / a problem
- Part III: Review of probability
  - Tail bounds

Part I: About the course

Info

- Webpage:
  http://www.cse.cuhk.edu.hk/~syzhang/course/MScAlg15
  - Information (time and venue, TA, textbook, etc.)
  - Lecture slides
  - Homework
  - Announcements

- Flavor:
  - More math than programming.

Homework

- Homework assignments (100%).
  - No exam.

- 12 homework.

- You only need to complete 10.
  - If you do more than 10, the 10 with the highest scores count.

textbook

- No textbook.
- Lecture notes available before classes.
- Some general references are listed in the course website as well.

Part II: About algorithms and complexity

A good example: driving directions

- Suppose we want to drive from CUHK to Central. How to route?
- Let’s ask Google.

- What's good here:
  - Step by step.
  - Each step is either turn left/right, or go straight for … meters.
  - An estimated time is also given.
- An algorithm is a computational procedure that has *step-by-step* instructions.
- It'll be good if an estimated time is given.

More on complexity

- Why time matters?
  - Time is money!
  - Being late means 0 value
    - Weather forecast.
    - Homework.
- Running time: the number of elementary steps
  - Assuming that each step only costs a small (or fixed) amount of time.

complexity

- The *worst-case time complexity* of an algorithm *A* is the running time of *A* on the *worst*-*case* input instance.
  - $\mathrm{Cost}(A) = \max_{\mathrm{input}\ x}(\text{running time of } A \text{ on } x)$

- The *worst-case time complexity* of a computational problem *P* is the worst-case complexity of the *best* algorithm *A* that solves the problem.
  - the best algorithm that gives right answers on all inputs.
  - $\mathrm{Cost}(P) = \min_{\mathrm{algorithm}\ A} \max_{\mathrm{input}\ x}(\text{running time of } A \text{ on } x)$

Hardness of problems can vary a lot

- Multiplication:
  - 1234 * 5678 = ?
    - 7006652
  - 2749274929483758 * 4827593028759302 = ?
    - Can you finish it in 10 minutes?
- Do you think you can handle multiplication easily?

Complexity of integer multiplication

- In general, for n-digit integers:
  - $x_1 x_2 \dots x_n * y_1 y_2 \dots y_n = ?$
- [Q] *How fast is our algorithm?*
- For each $y_i \quad (i = n, n-1, \dots, 1)$
  - we calculate $y_i * x_1 x_2 \dots x_n$,
    - n single-digit multiplications
    - n single-digit additions
- We finally add the n results (with proper shifts)
    - $\leq 2n^2$ single-digit additions.
- Altogether: $\leq 4n^2$ elementary operations
  - single-digit additions/multiplications
- Multiplication is not very hard even by hand, isn't it?

$$
\begin{array}{r}
x_1 x_2 \dots x_n \\
* \quad y_1 y_2 \dots y_n \\
\hline
* \; * \; * \; \dots * \\
* \; * \; * \; \dots * \quad \\
\dots \quad \dots \qquad \\
+ \quad * \; * \; * \; \dots * \qquad \quad \\
\hline
* \; * \; * \; * \; * \; * \dots \dots *
\end{array}
$$

Inverse problem

- The problem inverse to Integer Multiplication is Factoring.
- 35 = ? * ?
- 437?
- 8633?
- It’s getting harder and harder,
  - Much harder even with one more digit added!
- The best known algorithm: running time $\approx 2^{O(n^{1/3})}$

The bright side

- Hard problems can be used for cryptography!

- RSA [Rivest, Shamir, Adleman]:
  - widely-used today,
  - broken if one can factor quickly!

- One-line message: Quantum computers can factor quickly!

Messages

- Message 1: We care about the speed of the increase, especially when the size is very large.

- Many interesting instances in both theory and practice are of huge (and growing!) sizes.

- Message 2: We care about the big picture first.

- Is the problem as easy as multiplication, or as hard as factoring?

- In this regard, we consider the so called *asymptotic* behavior,…
  - Eventually, i.e. for large n, is the function like n, or n^2, or 2^n?
- with constant factors ignored at first
  - i.e. we care about the difference between n^2 and 2^n much more than that between n^2 and $1000n^2$
  - Engineering reason: speedup of a constant factor (say of 10) is easily achieved in a couple of years

Some examples

- Which increases faster?
  - $(100n^2, 0.01 * 2^n)$
  - $(0.1 * \log n, 10n)$
  - $(10^{10}n, 10^{-10}n^2)$

Big-O and small-o

- In general:
- $f(n) = O(g(n))$: for some constant c, $f(n) \leq c \cdot g(n)$, when n is sufficiently large.
  - i.e. $\exists c$, $\exists N$ s.t. $\forall n > N$, we have $f(n) \leq c \cdot g(n)$.
- $f(n) = o(g(n))$: for any constant c, $f(n) \leq c \cdot g(n)$, when n is sufficiently large.
  - i.e. $\forall c$, $\exists N$ s.t. $\forall n > N$, we have $f(n) \leq c \cdot g(n)$.

The other direction

- $f(n) = O(g(n))$: $f(n) \le c \cdot g(n)$ for some constant c and large n.
  - i.e. $\exists c$, $\exists N$ s.t. $\forall n > N$, we have $f(n) \le c \cdot g(n)$.
- $f(n) = \Omega(g(n))$: $f(n) \ge c \cdot g(n)$ for some constant c and large n.
  - i.e. $\exists c$, $\exists N$ s.t. $\forall n > N$, we have $f(n) \ge c \cdot g(n)$.
- $f(n) = \Theta(g(n))$: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
  - i.e. $c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$ for two constants c_1 and c_2 and large n.

Intuition

- $f = O(g)$
- $f = o(g)$
- $f = \Omega(g)$
- $f = \omega(g)$
- $f = \Theta(g)$

$\Leftrightarrow$

- $f \leq g$
- $f < g$
- $f \geq g$
- $f > g$
- $f = g$

- $f = O(g) \Leftrightarrow g = \Omega(f)$
- $f = o(g) \Leftrightarrow g = \omega(f)$
- $f = \Theta(g) \Leftrightarrow f = O(g)$ $\& \, f = \Omega(g)$

$\Leftrightarrow$

- $f \leq g \Leftrightarrow g \geq f$
- $f < g \Leftrightarrow g > f$
- $f = g \Leftrightarrow f \leq g \,\&\, f \geq g$

Examples

- $10n = o(0.1n^2)$
- $n^2 = o(2^{n/10})$
- $n^{1/3} = \omega(10 \log n)$

- $n^3 = (n^2)^{3/2} = \omega(n^2)$
- $\log_2 n^2 = 2 \log_2 n \; = \; \Theta(\log_2 n)$
- $\log_2(2n) = 1 + \log_2 n = \Theta(\log_2 n)$

Part III: Probability and tail bounds

Finite sample space

- Sample space Ω: set the all possible outcomes of a random process.
  - Suppose that Ω is finite.
- Events: subsets of Ω.
- Probability function. $p: \Omega \to R$, s.t.
  - $p(x) \geq 0, \forall x \in \Omega$.
  - $\sum_{x\in\Omega} p(x) = 1$.
- For event $E \subseteq \Omega$, the probability of event E happening is $p(E) = \sum_{x\in E} p(x)$.

Union of events

- Consider two events E_1 and E_2.
- $p(E_1 \cup E_2) = p(E_1) + p(E_2) - p(E_1 \cap E_2)$.

- In general, we have the following union bound:

$$p(\cup_i E_i) \leq \sum_i p(E_i)$$

Independence of events

- Two events A and B are independent if
$$p(A \cap B) = p(A)p(B)$$

- Conditional probability: For two events A and B with $p(B) > 0$, the *probability of* A *conditioned on* B is $p(A|B) = \frac{p(A \cap B)}{p(B)}$.

Random variable

- A random variable X is a function $X: \Omega \to R$.
- $\Pr[X = a] = \sum_{s \in \Omega : X(s) = a} p(s)$.
- Two random variables X and Y are independent if

$$\Pr[(X = a) \wedge (Y = b)] = \Pr[X = a] \Pr[Y = b].$$

Expectation

- Expectation:

$$\mathbf{E}[X] = \sum_{s\in\Omega} p(s)X(s)$$
$$= \sum_{i\in\mathrm{Range}(X)} i \cdot \Pr[X = i]$$

- Linearity of expectation:

$$\mathbf{E}\left[\sum_i X_i\right] = \sum_i \mathbf{E}[X_i]$$

no matter whether X_i's are independent or not.

variance

- The variance of X is
  $\mathbf{Var}[X] = \mathbf{E}[(X - \mathbf{E}[X])^2] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$

- The standard deviation of X is

$$\sigma = \sqrt{\mathbf{Var}[X]}$$

Concentration and tail bounds

- In many analysis of randomized algorithms, we need to study how concentrated a random variable X is close to its mean $E[X]$.
  - Many times $X = X_1 + \cdots + X_n$.
- Upper bounds of

$$\Pr[X \text{ deviates from } E[X] \text{ a lot}]$$

is called *tail bounds*.

Markov's Inequality: when you only know expectation

- [Thm] If $X \geq 0$, then

$$\mathbf{Pr}[X \geq a] \leq \frac{\mathbf{E}[X]}{a}.$$

In other words, if $E[X] = \mu$, then

$$\mathbf{Pr}[X \geq k\mu] \leq \frac{1}{k}.$$

- Proof. $\mathbf{E}[X] \geq a \cdot \mathbf{Pr}[X \geq a]$.
  - Dropping some nonnegative terms always make it smaller.

Moments

- Def. The k^{th} moment of a random variable X is

$$\mathbf{M}_k[X] \; = \; \mathbf{E}[(X - \mathbf{E}[X])^k]$$

- $k = 2$: variance.

$$\begin{aligned}\mathbf{Var}[X] &= \mathbf{E}[(X - \mathbf{E}[X])^2] \\ &= \mathbf{E}[X^2 - 2X \cdot \mathbf{E}[X] + \mathbf{E}[X]^2] \\ &= \mathbf{E}[X^2] - 2\mathbf{E}[X] \cdot \mathbf{E}[X] + \mathbf{E}[X]^2 \\ &= \mathbf{E}[X^2] - \mathbf{E}[X]^2\end{aligned}$$

Chebyshev's Inequality: when you also know variance

- [Thm] $\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a] \leq \frac{\mathbf{Var}[X]}{a^2}$.
  In other words,
  $$\mathbf{Pr}\left[|X - \mathbf{E}[X]| \geq k \cdot \sqrt{\mathbf{Var}[X]}\right] \leq \frac{1}{k^2}.$$
- Proof.
  $$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a]$$
  $= \mathbf{Pr}[|X - \mathbf{E}[X]|^2 \geq a^2]$
  $= \mathbf{Pr}[(X - \mathbf{E}[X])^2 \geq a^2]$
  $\leq \mathbf{E}[(X - \mathbf{E}[X])^2]/a^2$ // Markov on $(X - \mathbf{E}[X])^2$
  $= \mathbf{Var}[X]/a^2$ // recall: $\mathbf{Var}[X] = \mathbf{E}[(X - \mathbf{E}[X])^2]$

Inequality by the k^{th}-moment (k: even)

- [Thm] $\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a] \leq \mathbf{M}_k[X]/a^k$.
- Proof.

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq a]$$
$$= \mathbf{Pr}[|X - \mathbf{E}[X]|^k \geq a^k]$$
$$= \mathbf{Pr}[(X - \mathbf{E}[X])^k \geq a^k] \quad \text{// } k \text{ is even}$$
$$\leq \mathbf{E}[(X - \mathbf{E}[X])^k]/a^k \text{ // Markov on } (X - \mathbf{E}[X])^k$$
$$= \mathbf{M}_k[X]/a^k$$

Chernoff's Bound

- [Thm] Suppose $X_i = \begin{cases} 1 & \text{with prob. } p \\ 0 & \text{with prob. } 1-p \end{cases}$

  and let

  $$X = X_1 + \cdots + X_n.$$

  Then

  $$\mathbf{Pr}[|X - \mu| \geq \delta\mu] \leq e^{-\delta^2 \mu/3},$$

  where $\mu = np = \mathbf{E}[X]$.

Some basic applications

- One-sided error: Suppose an algorithm for a decision problem has
  - $f(x) = 0$: no error
  - $f(x) = 1$: output $f(x) = 0$ with probability 1/2
- We want to decrease this ½ to ε. How?
- Run the algorithm $\left\lceil \log_2\left(\frac{1}{\varepsilon}\right) \right\rceil$ times. Output 0 iff all executions answer 0.

Two-sided error

- Suppose a randomized algorithm has two-sided error
  - $f(x) = 0$: output $f(x) = 0$ with probability > 2/3
  - $f(x) = 1$: output $f(x) = 1$ with probability > 2/3

- How?
- Run the algorithm $O(\log(1/\varepsilon))$ steps and take a majority vote.

Using Chernoff's bound

- Run the algorithm n times, getting n outputs. Suppose they are $X_1, \dots, X_n$.
- Let $X = X_1 + \cdots + X_n$
  - if $f(x) = 0$: $X_i = 1$ *w.p.* $p < \frac{1}{3}$, thus $\mathbf{E}[X] = np < \frac{n}{3}$.
  - if $f(x) = 1$: $X_i = 1$ *w.p.* $p > \frac{2}{3}$, so $\mathbf{E}[X] = np > \frac{2n}{3}$.

- Recall Chernoff: $\mathbf{Pr}[|X-\mu| \geq \delta\mu] \leq e^{-\delta^2\mu/3}$.
- If $f(x)=0$: $\mu = \mathbf{E}[X] < \frac{n}{3}$.
  - $\delta\mu = \frac{n}{2} - \frac{n}{3} = \frac{n}{6}$, so $\delta = \frac{n/6}{n/3} = \frac{1}{2}$.
- $\mathbf{Pr}\left[X \geq \frac{n}{2}\right] \leq \mathbf{Pr}\left[|X-np| \geq \frac{n}{6}\right] \leq e^{-\frac{\delta^2\mu}{3}} = 2^{-\Omega(n)}$.
- Similar for $f(x)=1$.
- The error prob. decays exponentially with # of trials!