CS3230 Design and Analysis of Algorithms

AY2021/22 Semester 2

1. Introduction

1.1. Adversary Argument

- There are two different cases resulting in different outcomes but cannot be differentiated.

1.2. Minimum Step Problems

- No. of comparisons to find largest element: $n-1$
  - Second largest element: $n-1+\lg n-1$
- No. of comparisons in sorting algorithm: $n\lg n$
- No. of edges checked to tell connectivity: $\binom{n}{2}$

2. Asymptotic Analysis

2.1. Asymptotic Notations

Notation	Definition	$\lim_{n\to\infty}\frac{f(n)}{g(n)}$
$f(n)=O(g(n))$	$\exists c>0, n_0>0$ s.t. $\forall n\geq n_0, 0\leq f(n)\leq cg(n)$	$<\infty$
$f(n)=o(g(n))$	$\forall c>0, \exists n_0>0$ s.t. $\forall n\geq n_0, 0\leq f(n)<cg(n)$	$=0$
$f(n)=\Omega(g(n))$	$\exists c>0, n_0>0$ s.t. $\forall n\geq n_0, o\leq cg(n)\leq f(n)$	>0
$f(n)=\omega(g(n))$	$\forall c>0, \exists n_0>0$ s.t. $\forall n\geq n_0, 0\leq cg(n)<f(n)$	$=\infty$
$f(n)=\Theta(g(n))$	$\exists c_1, c_2>0, n_0>0$ s.t. $\forall n\geq n_0, c_1 g(n)\leq f(n)\leq c_2 g(n)$	$(0,\infty)$

2.2. Useful Facts

- $\forall k,d>0, (\lg n)^k = o(n^d)$
- $\forall d>0, u>1, n^d = o(u^n)$
- Stirling's Formula: $n! \approx \left(\frac{n}{e}\right)^n\sqrt{2\pi n}$
  - $\lg(n!) = \Theta(n\lg n)$
  - $\lg\lg n + \lg\lg\frac{n}{2} + \lg\lg\frac{n}{4} + \cdots + 1 = \lg(\lg n!) = \lg n\lg\lg n$
- Harmonic Series: $1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n} = \Theta(\lg n)$
- Decision Tree with n variables (e.g. sorting):
  - $h = \Omega(\lg(n!)) = \Omega(n\lg n)$
- $\lim_{n\to\infty}\left(\frac{n-1}{n}\right)^n = \frac{1}{e}$

3. Iteration, Recursion and Divide and Conquer

3.1. Correctness of Iterative Algorithm (Loop Invariant)

- Initialisation: The invariant is true before the first iteration of the loop.
- Maintenance: If the invariant is true before an iteration, it remains true before the next iteration.
- Termination: When the algorithm terminates, the invariant provides a useful property for showing correctness.

3.2. Correctness of Recursive Algorithm (Strong Induction)

- Prove base cases.
- Assuming the algorithm works for smaller cases, show that it works correctly.

3.3. Solve a Recurrence

- Recursion Tree: Draw the recursion tree and count total number of operations.
- Master Method for $T(n) = aT\left(\frac{n}{b}\right) + \Theta(f(n))$:

Condition	Solution
$f(n) = O(n^{\log_b a-\epsilon})$ for some $\epsilon>0$	$T(n)=\Theta(n^{\log_b a})$
$f(n) = \Theta(n^{\log_b a}\log^k n)$ for some $k\geq 0$	$T(n)=\Theta(n^{\log_b a}\log^{k+1} n)$
$f(n) = \Omega(n^{\log_b a+\epsilon})$ for some $\epsilon>0$ $af\left(\frac{n}{b}\right)\leq cf(n)$ for some $c<1$	$T(n) = \Omega(f(n))$

- Substitution Method:
  - Guess the form of the solution.
  - Verify by induction.

Example: Solve $T(n) = 4T\left(\frac{n}{2}\right)+n$.

1. Guess $T(n)=O(n^2)$. Assume $T(1)=q$.
2. We are to show that $\exists c_1, c_2>0, n_0>0$ s.t. $\forall n\geq n_0, 0\leq T(n)\leq c_1 n^2 - c_2 n$.
3. Set $c_1 = q+1, c_2 = 1, n_0 = 1$.
4. Base case: $T(1) = q \leq (q+1)-1$.
5. Recursive case:
$$T(n) = 4T\left(\frac{n}{2}\right)+n \leq 4\left(c_1\cdot\frac{n^2}{4} - c_2\cdot\frac{n}{2}\right)+n = n^2 - n = c_1 n^2 - c_2 n$$

4. Average Case Analysis and Randomised Algorithms

4.1. MergeSort vs QuickSort

- MergeSort is more efficient theoretically, but QuickSort is preferred empirically.
  - MergeSort requires extra memory.
  - Cache misses.
- Colin McDiarmid Theorem: Probability that the runtime of Randomised QuickSort exceeds average by $x\% = n^{-\frac{x}{100}\ln\ln n}$.

4.2. Geometric Distribution

- Suppose $X\sim Geo(p)$, then $E(X)=\frac{1}{p}$.

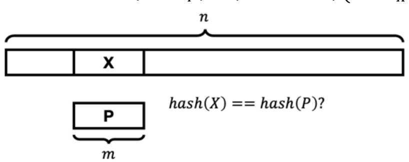
5. Hashing

5.1. Universal Hashing

- Suppose $\mathcal{H}$ is a set of hash functions mapping U to $[M]$. We say $\mathcal{H}$ is universal if:
$$\forall x\neq y, \frac{|h\in\mathcal{H} : h(x)=h(y)|}{|\mathcal{H}|}\leq\frac{1}{M}$$
That is, if we choose h randomly from $\mathcal{H}$, then for any $x\neq y$, the probability of them having the same hash value is smaller than or equal to $\frac{1}{M}$.
- Indicator Variable: $X = \begin{cases}1, \text{if event } A \text{ occurs}\\ 0, \text{if event } A \text{ does not occur}\end{cases}$
  - For n elements, the expected number of collisions between any pair of them is:
$$\leq\binom{n}{2}\cdot\frac{1}{M}$$

5.2. Karp-Rabin Algorithm

- Faster string equality:
Total runtime for pattern matching
$$= |hash_P| + (n-m+1)(hash_X + O(1))$$



- Rolling hash – Division Hash:

1. Choose p to be a random prime number in the range $\{1,\dots,K\}$.
2. Define, for any integer x, $h_p(x) = x \bmod p$.

  - Useful fact: Number of prime numbers in $\{1,\dots,K\} > K/\ln K$.
  - Hence, if $0\leq x<y<2^b$, then
$$\Pr(h_p(x)==h_p(y)) < \frac{b\ln K}{K}$$
  - Set $K = 200mn\ln(200mn)$. Then the probability of getting a false positive is $<$ 1%.
  - Roll from $T[1,\dots,m]$ to $T[2,\dots,m+1]$:
$$h_p(X') = \left(h_p(X) - T[1]\cdot h_p(2^{m-1})\right)\cdot 2 + T[m+1] \pmod p$$
- Monte-Carlo Algorithm

1. Pick random prime p from $\{1, \lceil 200mn\ln 200mn\rceil\}$.
2. Compute $h_p(P)$, $h_p(2^m)$ and $h_p(T[1,\dots,m])$.
3. Check if $h_p(P) == h_p(T]1,\dots,m])$.
4. Start rolling and check each substring equality.

  - Runtime: $O(m+n)$
  - Error probability: $<1\%$