CS3230
AY21/22 SEM 2
github/jovyntls

01. COMPUTATIONAL MODELS

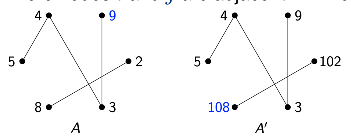
- **algorithm** $\rightarrow$ a well-defined procedure for finding the correct solution to the input
- **correctness**
  - **worst-case correctness** $\rightarrow$ correct on *every valid input*
  - other types of correctness: correct on random input/with high probability/approximately correct
- **efficiency** / **running time** $\rightarrow$ measures the number of steps executed by an algorithm as a function of the *input size* (depends on computational model used)
  - number input: typically the length of binary representation
  - **worst-case** running time $\rightarrow$ *max* number of steps executed when run on an input of size n

Comparison Model

- algorithm can **compare** any two elements in one time unit ($x > y, x < y, x = y$)
- running time = number of comparisons made
- array can be manipulated at no cost

Maximum Problem

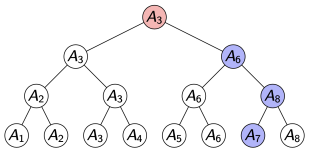
- *problem*: find the largest element in an array A of n distinct elements
- *proof that $n-1$ comparisons are needed*:
  - fix an algorithm M that solves the Maximum problem on all inputs using $< n-1$ comparisons. construct graph G where nodes i and j are adjacent iff M compares i and j.



  - M cannot differentiate A and A'.
- **adversary argument** $\rightarrow$ inputs are decided such that they have different solutions

Second Largest Problem

- *problem*: find the second largest element in $< 2n-3$ comparisons (2x Maximum $\Rightarrow (n-1)+((n-1)-1)=2n-3$)
- *solution*: **knockout tournament** $\Rightarrow n + \lceil \lg n \rceil - 2$



1. bracket system: $n-1$ matches
   - every non-winner has lost exactly once
2. then compare the elements that have lost to the largest
   - the second-largest element must have lost to the winner
   - compares $\lceil \lg n \rceil$ elements that have lost to the winner using $\lceil \lg n \rceil - 1$ comparisons

Sorting

- there is a sorting algorithm that requires $\leq n \lg n - n + 1$ comparisons.
- *proof:* every sorting algorithm must make $\geq \lg(n!)$ comparisons.
  1. let set $\mathcal{U}$ be the set of all permutations of the set $\{1, \ldots, n\}$ that the adversary could choose as array A. $|\mathcal{U}| = n!$
  2. for each query "is $A_i > A_j$?", if $\mathcal{U}_{yes} = \{A \in \mathcal{U} : A_i > A_j\}$ is of size $\geq |\mathcal{U}|/2$, set $\mathcal{U} := \mathcal{U}_{yes}$. else: $\mathcal{U} := \mathcal{U} \backslash \mathcal{U}_{yes}$
  3. the size of $\mathcal{U}$ decreases by at most half with each comparison
  4. for $> \lg(n!)$ comparisons, $\mathcal{U}$ will still contain at least 2 permutations

$$n! \geq (\tfrac{n}{e})^n$$
$$\Rightarrow \lg(n!) \geq n \lg(\tfrac{n}{e}) = n \lg n - n \lg e$$
$$\approx n \lg n - 1.44n$$

$\Rightarrow$ roughly $n \lg n$ comparisons are **required** and **sufficient** for sorting n numbers

String Model

- input: string of n bits
- each query: find out **one bit** of the string
- n queries are **necessary** and **sufficient** to check if the input string is all 0s.

Graph Model

- input: (symmetric) adjacency matrix of an n-node undirected graph
- each query: find out if an edge is present between two chosen nodes
- *proof*: $\binom{n}{2}$ queries are necessary to decide whether the graph is connected or not
  1. suppose M is an algorithm making $\leq \binom{n}{2}$ queries.
  2. whenever M makes a query, the algorithm tries not adding this edge, but adding all remaining unqueried edges.
     2.1. if the resulting graph is connected, M replies 0 (i.e. edge does not exist)
     2.2. else: replies 1 (edge exists)
  3. after $< \binom{n}{2}$ queries, at least one entry of the adjacency matrix is unqueried.

02. ASYMPTOTIC ANALYSIS

- **algorithm** $\rightarrow$ a *finite* sequence of well-defined instructions to solve a given computational problem
- **runtime** $\rightarrow$ measured in number of instructions taken in **word-RAM** model
  - operators, comparisons, if, return, etc

Asymptotic Notations

upper bound ($\leq$): $f(n) = O(g(n))$
if $\exists c > 0, n_0 > 0$ such that $\forall n \geq n_0, \quad 0 \leq f(n) \leq cg(n)$

lower bound ($\geq$): $f(n) = \Omega(g(n))$
if $\exists c > 0, n_0 > 0$ such that $\forall n \geq n_0, \quad 0 \leq cg(n) \leq f(n)$

tight bound: $f(n) = \Theta(g(n))$
if $\exists c_1 > 0, c_2 > 0, n_0 > 0$ such that $\forall n \geq n_0, \quad 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$

o **notation** ($<$): $f(n) = o(g(n))$
if $\forall c > 0, \exists n_0 > 0$ such that $\forall n \geq n_0$, $0 \leq f(n) < cg(n)$

ω**-notation** ($>$): $f(n) = \omega(g(n))$
if $\forall c > 0, \exists n_0 > 0$ such that $\forall n \geq n_0$, $0 \leq cg(n) < f(n)$

Set definitions

- *upper*: $O(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \mid \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$
- *lower*: $\Omega(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \mid \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$

Proof. that $2n^2 = O(n^3)$
let $f(n) = 2n^2$. then $f(n) = 2n^2 \leq n^3$ when $n \geq 2$.
set $c = 1$ and $n_0 = 2$.
we have $f(n) = 2n^2 \leq c \cdot n^3$ for $n \geq n_0$. $\square$

Limits

Assume $f(n), g(n) > 0$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = O(g(n))$$
$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = \Theta(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \Rightarrow f(n) = \Omega(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) = \omega(g(n))$$

Proof. using delta epsilon definition

Properties of Big O

$$\boxed{\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))}$$

- **transitivity** - applies for $O, \Theta, \Omega, o, \omega$
  $f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
- **reflexivity** - for $O, \Omega, \Theta, \quad f(n) = O(f(n))$
- **symmetry** - $f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$
- **complementarity** -
  - $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$
  - $f(n) = o(g(n)) \iff g(n) = \omega(f(n))$

insertion sort: $O(n^2)$ with worst case $\Theta(n^2)$

$$\boxed{\log \log n < \log n < (\log n)^k < n^k < k^n}$$

03. ITERATION, RECURSION, DIVIDE-AND-CONQUER

Iterative Algorithms

loop invariant implies correctness if
- *initialisation* - true before the first iteration
- *maintenance* - if true before an iteration, remains true at the beginning of the next iteration
- *termination* - true at the end

Divide-and-Conquer

powering a number

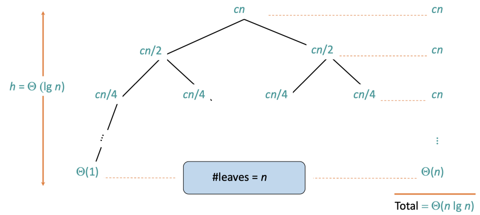
- problem: compute $f(n,m) = a^n \pmod m$ for all integer n, m
- observation: $f(x+y, m) = f(x,m) * f(y,m) \pmod m$
- **naive solution**: recursively compute and combine $f(n-1,m) * f(1,m) \pmod m$
  - $T(n) = T(n-1) + T(1) + \Theta(1) \Rightarrow T(n) = \Theta(n)$
- **better solution**: divide and conquer
  - divide: trivial
  - conquer: recursively compute $f(\lfloor n/2 \rfloor, m)$
  - combine:
    - $f(n,m) = f(\lfloor n/2 \rfloor, m)^2 \pmod m$ if n is even
    - $f(n,m) = f(1,m) * f(\lfloor n/2 \rfloor, m)^2 \pmod m$ if odd
  - $T(n) = T(n/2) + \Theta(1) \Rightarrow \Theta(\log n)$

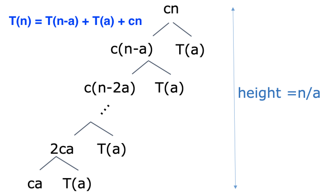
Solving Recurrences

for a sub-problems of size $\frac{n}{b}$ where $f(n)$ is the time to divide and combine, $T(n) = aT(\frac{n}{b}) + f(n)$

Recursion tree

total = height $\times$ number of leaves





Master method

$T(n) = aT(\frac{n}{b}) + f(n)$
$a \geq 0, b > 1, f$ is asymptotically positive

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } f(n) < n^{\log_b a} \text{ polynomially} \\ \Theta(n^{\log_b a} \log n) & \text{if } f(n) = n^{\log_b a} \\ \Theta(f(n)) & \text{if } f(n) > n^{\log_b a} \text{ polynomially} \end{cases}$$

harmonic series: $\sum_{k=1}^{\infty} \frac{1}{k} \approx \ln k = \Theta(\lg n)$

Substitution method

1. guess that $T(n) = O(f(n))$.
   i.e. $\exists c$ such that $T(n) \leq c \cdot f(n)$, for $n \geq n_0$.
2. verify by induction:
   2.1. set $c = \max\{2, q\}$ and $n_0 = 1$
   2.2. base case ($n = n_0 = 1$)
   2.3. recursive case ($n > 1$):
   - by strong induction, assume $T(k) = c \cdot f(k)$ for $n > k > 1$
   - T(n) = <recurrence> ... $\leq c \cdot f(n)$

   2.4. hence $T(n) \leq c \cdot f(n)$ for $n \geq 1$.

! may not be a tight bound!

example

$T(n) = 4T(n/2) + n^2/\lg n \Rightarrow \Theta(n^2 \lg \lg n)$

Proof. $T(n) = 4T(n/2) + \frac{n^2}{\lg n}$
$= 4(4T(n/4) + \frac{(n/2)^2}{\lg n - lg 2}) + \frac{n^2}{\lg n}$
$= 16T(n/4) + \frac{n^2}{\lg n - \lg 2} + \frac{n^2}{\lg n}$
$= \sum_{k=1}^{\lg n} \frac{n^2}{\lg n - k}$
$= n^2 \lg \lg n$ by approx. of harmonic series ($\sum \frac{1}{k}$)

04. AVERAGE-CASE ANALYSIS & RANDOMISED ALGORITHMS

Quicksort Analysis

- divide & conquer, linear-time $\Theta(n)$ partitioning subroutine
- assume we select the first array element as pivot
- if the pivot produces subarrays of size j and $(n-j-1)$, then $T(n) = T(j) + T(n-j-1) + \Theta(n)$

time analysis

- **worst-case**: $T(n) = T(0) + T(n-1) + \Theta(n) \Rightarrow \Theta(n^2)$
- **average case** $A(n) \rightarrow$ expected running time when the input is chosen uniformly at random from the set of all $n!$ permutations
- **average case**, $A(n) = \frac{1}{n!}\sum_\pi Q(\pi)$ where $Q(\pi)$ is the time complexity when the input is permutation π.

Proof. for quicksort, $A(n) = O(n \log n)$

let $P(i)$ be the set of all those permutations of elements $\{e_1, e_2, \dots, e_n\}$ that begins with e_i.
Let $G(n,i)$ be the average running time of quicksort over $P(i)$. Then
$G(n) = A(i-1) + A(n-i) + (n-1)$.
$A(n) = \frac{1}{n}\sum_{i=1}^n G(n,i)$
$= \frac{1}{n}\sum_{i=1}^n (A(i-1) + A(n-i) + (n-1))$
$= \frac{2}{n}\sum_{i=1}^n A(i-1) + n - 1$
$= O(n \log n)$ by taking it as area under integration

quicksort vs mergesort

	average	best	worst
quicksort	$1.39 n \lg n$	$n \lg n$	$n(n-1)$
mergesort	$n \lg n$	$n \lg n$	$n \lg n$

- disadvantages of mergesort:
  - overhead of temporary storage
  - cache misses
- advantages of quicksort
  - in place
  - reliable (as $n \uparrow$, chances of deviation from avg case $\downarrow$)
- issues with quicksort
  - **distribution-sensitive** $\rightarrow$ time taken depends on the initial (input) permutation

Randomised Algorithms

- **randomised algorithms** $\rightarrow$ output and running time are **functions** of the **input** and **random bits chosen**
  - vs non-randomised: output & running time are functions of the *input only*
- **randomised quicksort**: choose pivot at random
  - probability that the runtime of *randomised* quicksort exceeds average by $x\% = n^{-\frac{x}{100}\ln\ln n}$
  - P(time takes at least double of the average) = 10^{-15}
  - distribution insensitive

Randomised Quicksort Analysis

$T(n) = n - 1 + T(q-1) + T(n-q)$
Let $A(n) = \mathbb{E}[T(n)]$ where the expectation is over the randomness in expectation.
Taking expectations and applying linearity of expectation:
$A(n) = n - 1 + \frac{1}{n}\sum_{q=1}^n (A(q-1) + A(n-q))$
$= n - 1 + \frac{2}{n}\sum_{q=1}^{n-1} A(q)$
$A(n) = n \log n \quad \Rightarrow$ same as average case quicksort

Randomised Quickselect

- $O(n)$ to find the k^{th} smallest element
- randomisation: unlikely to keep getting a bad split

Types of Randomised Algorithms

- randomised **Las Vegas** algorithms
  - output is always correct
  - runtime is a *random variable*
  - e.g. randomised quicksort
- randomised **Monte Carlo** algorithms
  - output may be incorrect with some small probability
  - runtime is *deterministic*

examples

- *smallest enclosing circle:* given n points in a plane, compute the smallest radius circle that encloses all n points
  - best **deterministic** algorithm: $O(n)$, but complex
  - las vegas: average $O(n)$, simple solution
- *minimum cut:* given a connected graph G with n vertices and m edges, compute the smallest set of edges whose removal would disconnect G.
  - best **deterministic** algorithm: $O(mn)$
  - **monte carlo**: $O(m \log n)$, error probability n^{-c} for any c
- *primality testing:* determine if an n bit integer is prime
  - best **deterministic** algorithm: $O(n^6)$
  - **monte carlo**: $O(kn^2)$, error probability 2^{-k} for any k

Geometric Distribution

Let X be the number of trials repeated until success.
X is a random variable and follows a geometric distribution with probability p.

Expected number of trials, $E[X] = \frac{1}{p}$
$Pr[X = k] = q^{k-1}p$

Linearity of Expectation

For any two events X, Y and a constant a,
$E[X+Y] = E[X] + E[Y]$
$E[aX] = aE[X]$

Coupon Collector Problem

n types of coupon are put into a box and randomly drawn with replacement. What is the expected number of draws needed to collect at least one of each type of coupon?

- let T_i be the time to collect the i-th coupon after the $i-1$ coupon has been collected.
  - Probability of collecting a new coupon, $p_i = \frac{(n-(i-1))}{n}$
  - T_i has a **geometric distribution**
  - $E[T_i] = 1/p_i$
- total number of draws, $T = \sum_{i=1}^n T_i$
- $E[T] = E[\sum_{i=1}^n T_i] = \sum_{i=1}^n E[T_i]$ by linearity of expectation
  $= \sum_{i=1}^n \frac{n}{n-(i-1)} = n \cdot \sum_{i=1}^n \frac{1}{i} = \Theta(n \lg n)$

05. HASHING

Dictionary ADT

- different types:
  - **static** - fixed set of inserted items; only care about queries
  - **insertion-only** - only insertions and queries
  - **dynamic** - insertions, deletions, queries
- implementations
  - sorted list (static) - $O(\log N)$ query
  - balanced search tree (dynamic) - $O(\log N)$ all operations
  - direct access table
    - $\times$ needs items to be represented as non-negative integers (**prehashing**)
    - $\times$ huge space requirement
- using $\mathcal{H}$ for dictionaries: need to store both the hash table and the matrix A.
  - additional storage overhead $= \Theta(\log N \cdot \log |U|)$, if $M = \Theta(N)$
  - other universal hashing constructions may have more efficient hash function evaluation

Hashing

- **hash function**, $h : U \rightarrow \{1, \dots, M\}$ gives the location of where to store in the hash table
  - notation: $[M] = \{1, \dots, M\}[M] = \{1, \dots, M\}$
- **collision** $\rightarrow$ for two different keys x and y, $h(x) = h(y)$
  - resolve by **chaining**, **open addressing**, etc
- desired properties
  - ✓ minimise collisions - `query(x)` and `delete(x)` take time $\Theta(|h(x)|)$
  - ✓ minimise storage space - aim to have $M = O(N)$
  - ✓ function h is easy to compute (assume constant time)
- if $|U| \geq (N-1)M + 1$, for any $h : U \rightarrow [M]$, there is a set of N elements having the same hash value.
  - *Proof*: pigeonhole principle
- use **randomisation** to overcome the adversary
  - e.g. randomly choose between two *deterministic* hash functions h_1 and h_2
    $\Rightarrow$ for any pair of keys, with probability $\geq \frac{1}{2}$, there will be no collision

Universal Hashing

Suppose $\mathcal{H}$ is a set of hash functions mapping U to $[M]$.

$\mathcal{H}$ is **universal** if $\forall x \neq y$, $\frac{|h \in \mathcal{H} : h(x) = h(y)|}{|H|} \leq \frac{1}{M}$
or $\underset{h \sim \mathcal{H}}{Pr}[h(x) = h(y)] \leq \frac{1}{M}$

- aka: for any $x \neq y$, if h is chosen uniformly at random from a universal $\mathcal{H}$, then there is at most $\frac{1}{M}$ probability that $h(x) = h(y)$
- probability where h is sampled uniformly from $\mathcal{H}$

Collision Analysis

Suppose $\mathcal{H}$ is a *universal* family of HFs mapping U to $[M]$.

For any N elements $x_1, x_2, \dots, x_N$, the **expected number of collisions** between x_N and the other elements is $< \frac{N}{M}$.

Proof. let A_i be an indicator r.v. for $h(x_i) = h(x_N)$.
$E[A_i] = 1 \cdot P(A_i = 1) + 0 \cdot P(A_i = 0) = P(A_i = 1) \leq \frac{1}{M}$.
# of collisions with x_N is $\sum_{i<N} A_i$

Expected Cost

Suppose $\mathcal{H}$ is a *universal* family of HFs mapping U to $[M]$.

For any sequence of N insertions, deletions and queries, if $M \geq N$, then the **expected total cost** for a random $h \in \mathcal{H}$ is $O(N)$.

Proof. Each operation costs $O(1)$ time by this claim.
Linearity of expectation $\Rightarrow$ total $O(N)$

Construction of Universal Family

Obtain a universal family of hash functions with $M = O(N)$.

- Suppose U is indexed by u-bit strings and $M = 2^m$.
- For any $m \times u$ binary matrix A, $h_A(x) = Ax \pmod 2$
  - each element `x => x % 2`
  - x is a $u \times 1$ matrix $\Rightarrow Ax$ is $m \times 1$
- *Claim*: $\{h_A : A \in \{0,1\}^{m \times u}\}$ is universal
- e.g. $U = \{00, 01, 10, 11\}, M = 2$
  - h_{ab} means $A = [a\ b]$

	00	01	10	11
h_{00}	0	0	0	0
h_{01}	0	1	0	1
h_{10}	0	0	1	1
h_{11}	0	1	1	0

Proof. Let $x \neq y$. Let $z = x - y$. We know $z \neq 0$.
Collision: $P(Ax = Ay) = P[A(x-y) = 0] = P(Az = 0)$.
To show $P(Az = 0) \leq \frac{1}{M}$.
Special case - Suppose z is 1 at the i-th coordinate but 0 everywhere else. Then Az is the i-th column of A.
Since the i-th column is uniformly random,
$P(Az = 0) = \frac{1}{2^m} = \frac{1}{M}$.
General case - Suppose z is 1 at the i-th coordinate.
Let $z = [z_1\ z_2\ \dots\ z_u]^T$. $A = [A_1\ A_2\ \dots\ A_u]$
hence A_k is the k-th column of A.
Then $Az = z_1A_1 + z_2A_2 + \cdots + z_uA_u$.
$Az = 0 \Rightarrow z_1A_1 = -(z_2A_2 + \cdots + z_uA_u) \quad (*)$
We fix z_1A_1 to be an arbitrary $m \times 1$ matrix of 1s and 0s. The probability that $(*)$ holds is $\frac{1}{2^m}$.

Perfect Hashing

static case - N fixed items in the dictionary $x_1, x_2, \dots, x_N$

Quadratic Space

if $\mathcal{H}$ is universal and $M = N^2$, and h is sampled uniformly from $\mathcal{H}$, then the expected number of collisions is < 1.

Proof. for $i \neq j$, let indicator r.v. A_{ij} be equal to 1 if $h(x_i) = h(x_j)$, or 0 otherwise.
By universality, $E[A_{ij}] = P(A_{ij} = 1) \leq 1/N^2$
$E[\text{\# collisions}] = \sum_{i<j} E[A_{ij}] \leq \binom{N}{2}\frac{1}{N^2} < 1$

2-Level Scheme

No collision and less space needed

Construction

Choose $h : U \to [N]$ from a universal hash family.

- Let L_k be the number of x_i's for which $h(x_i) = k$.
- Choose $h_1, \dots, h_N$ **second-level** hash functions $h_k : [N] \to [(L_k)^2]$ s.t. there are no collisions among the L_k elements mapped to k by h.
  - quadratic second-level table $\to$ ensures no collisions using quadratic space

Analysis

if $\mathcal{H}$ is universal and h is sampled uniformly from $\mathcal{H}$, then

$$E\left[\sum_k L_k^2\right] < 2N$$

Proof. For $i, j \in [1, N]$, define indicator r.v. $A_{ij} = 1$ if $h(x_i) = h(x_j)$, or 0 otherwise.

$A_{ij} =$ # possible collisions = # pairs * 2 $= L_k^2$
Hence $\sum_k L_k^2 = \sum_{i,j} A_{ij}$

$$\begin{aligned} E[\sum_{i,j} A_{ij}] &= \sum_i E[A_{ii}] + \sum_{i \neq j} E[A_{ij}] \\ &\leq N \cdot 1 + N(N-1) \cdot \frac{1}{N} \\ &< 2N \end{aligned}$$

helpful approximations

harmonic number, $H_n = \sum_{k=1}^{n} \frac{1}{k} = \Theta(\lg n)$