CS3230 Cheatsheet by MA

Stable Matching

A matching is **stable** if no unmatched man and woman both prefer each other to their current partners, ie. Gale-Shapley Algorithm

Asymptotic Analysis

1. O**-notation** (upper bound)
$O(g(n)) = \{f(n)$: there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0\}$
2. Θ**-notation** (tight bound)
$\Theta(g(n)) = \{f(n)$: there exist constants $c_1 > 0$, $c_2 > 0$, $n_0 > 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0\}$
3. Ω**-notation** (lower bound)
$\Omega(g(n)) = \{f(n)$: there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0\}$
4. o**-notation** (tight upper bound)
$o(g(n)) = \{f(n)$: for any constant $c > 0$, there is a constant $n_0 > 0$ such that $0 \leq f(n) < c \cdot g(n)$ for all $n \geq n_0\}$
5. ω**-notation** (tight lower bound)
$\omega(g(n)) = \{f(n)$: for any constant $c > 0$, there is a constant $n_0 > 0$ such that $0 \leq c \cdot g(n) < f(n)$ for all $n \geq n_0\}$

Properties of big-O

Transitivity

$$\begin{aligned}
f(n) = \Theta(g(n)) \;\; & \& \;\; g(n) = \Theta(h(n)) \\
& \Rightarrow f(n) = \Theta(h(n)) \\
f(n) = O(g(n)) \;\; & \& \;\; g(n) = O(h(n)) \\
& \Rightarrow f(n) = O(h(n)) \\
f(n) = \Omega(g(n)) \;\; & \& \;\; g(n) = \Omega(h(n)) \\
& \Rightarrow f(n) = \Omega(h(n)) \\
f(n) = o(g(n)) \;\; & \& \;\; g(n) = o(h(n)) \\
& \Rightarrow f(n) = o(h(n)) \\
f(n) = \omega(g(n)) \;\; & \& \;\; g(n) = \omega(h(n)) \\
& \Rightarrow f(n) = \omega(h(n))
\end{aligned}$$

Reflexivity

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

Symmetry

$$f(n) = \Theta(g(n)) \text{ iff } g(n) = \Theta(f(n))$$

Complementary

$$f(n) = O(g(n)) \text{ iff } g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \text{ iff } g(n) = \omega(f(n))$$

Properties of Math

Exponential

$$a^{-1} = 1/a$$
$$(a^m)^n = a^{mn}$$
$$a^m a^n = a^{m+n}$$
$$e^x \geq 1 + x$$

Logarithm

$$\lg n = \log_2 n$$
$$\ln n = \log_e n$$
$$\lg^k n = (\lg n)^k$$
$$lg\, lg\, n = lg(lg\, n))$$
$$a = b^{\log_b a}$$
$$\log_c(ab) = \log_c a + \log_c b$$
$$\log_b a^n = n \log_b a$$
$$\log_b a = \frac{\log_c a}{\log_c b}$$
$$\log_b(1/a) = -\log_b a$$
$$\log_b a = \frac{1}{\log_a b}$$
$$a^{\log_b c} = c^{\log_b a}$$

Stirling's approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \theta\left(\frac{1}{n}\right)\right)$$
$$\log(n!) = \theta(n \lg n)$$

Arithmetic Series

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + ... + n$$
$$= \frac{1}{2}n(n+1) = \Theta(n^2)$$

Geometric Series

$$\sum_{k=1}^{n} x^k = 1 + x^2 + x^3 + ... + x^n = \frac{x^{n+1} - 1}{x - 1}$$
$$\sum_{k=1}^{\infty} x^k = \frac{1}{1-x} \text{ when } |x| < 1$$

Harmonic Series

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$$
$$= \sum_{k=1}^{n} \frac{1}{k}$$
$$= \ln n + O(1)$$

Telescoping Series

For any sequence $a_0, a_1, ..., a_n$

$$\sum_{k=0}^{n-1}(a_k - a_{k+1}) = \begin{matrix} (a_0 - \cancel{a_1}) + \\ (\cancel{a_1} - \cancel{a_2}) + \\ (\cancel{a_2} - \cancel{a_3}) + \\ \ldots \\ (\cancel{a_{n-1}} - a_n) \end{matrix} = a_0 - a_n$$

Limit

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

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

L'Hopital's Rule

If we have an indeterminate form $\frac{0}{0}$ or $\frac{\infty}{\infty}$, then

$$\lim_{x\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{x\to\infty}\left(\frac{f'(n)}{g'(n)}\right)$$

Correctness of Algorithms

Iterative Algorithms

A **loop invariant** is:

- true at the beginning of an iteration, and
- remains true at the beginning of the next iteration
- if true at the end, then it implies algorithm's correctness

To use invariant to show the correctness of an iterative algorithm, we need to show three things:

- **Initialization:** 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.

Recursive Algorithms

To show the correctness of a recursive algorithm:

- Use strong induction
- Prove base cases
- Show algorithm works correctly assuming algorithm works correctly for smaller cases.

Solve Recurrences

Recursion tree

Draw out the recurrence in the form of a tree

Master method

Master theorem applies to recurrences of the form

$$T(n) = a\,T(n/b) + f(n)$$

where $a \geq 1, b > 1$ and f is asymptotically positive

When comparing $f(n)$ and $n^{\log_b a}$ There are three cases to master theorem.
Define a, b, $f(n)$ and $n^{log_b a}$

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   $T(n) = \Theta(n^{\log_b a})$
2. $f(n) = \Theta(n^{\log_b a} \log^k n)$ for some constant $k \geq 0$.
   $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$ **and** $f(n)$ satisfies the **regularity condition** that
   $af(n/b) \leq cf(n)$ for some constant $c < 1$.
   $T(n) = \Theta(f(n))$

Substitution method

Guess the time complexity and verify that it is correct by induction

Telescoping method

Expand out the recurrence, until the base case then add up

Randomized Algorithms

An algorithm is called **randomized** if its output and running time are determined by its input and also by values produced by a random-number generator.

Random Variables and Expectation

A **discrete random variable** is a function from a sample space to the integers.
Expectation: $E[X] = \sum_i i \cdot Pr[X = i]$
Linearity of expectations: $E[X + Y] = E[X] + E[Y]$ for any two random variables X and Y.

Examples of Randomized Algorithms

Monte Carlo Algorithm: Randomized algorithm that gives the correct answer with probability 1 - o(1) ("high probability"), but run-time bound holds deterministically

- finding π by randomly sampling n (x,y) and count fractions satisfying $x^2 + y^2 \leq 1$ then multiply by 4 to get an estimate to π
- run-time is $\Theta(n)$ but only approximates

Las Vegas Algorithm: Randomized algorithm that always gives the correct answer, but the run-time is a random variable

- very simple
- average $O(n)$ time complexity

Hashing

SUHA: $p(h(x_i) = h(x_j)) \leq \frac{1}{m}$
Different types of dictionaries:
Static: set of inserted items fixed; only care about queries
Insertion only: only insertions and queries
Dynamic: insertions, deletions and queries

Universal Hashing

Suppose H is a set of hash functions mapping U to $[M]$. We say H is **universal** if for all $x \neq y$:

$$\frac{|h \in H : h(x) = h(y)|}{|H|} \leq \frac{1}{M}$$

For any $x \neq y$, if h is chosen uniformly at random from a universal H, there's at most $\frac{1}{M}$ probability that $h(x) = h(y)$.

Collision Analysis

Suppose H is a *universal* family of hash functions mapping U to $[M]$. For any N elements, $x_1, ..., x_N$, the expected number of collisions between x_N and the other elements is $< \frac{N}{M}$.

Expected Cost

Suppose H is a *universal* family of hash functions 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 $h \in H$ is $O(n)$.

Construction of universal family

Suppose U is indexed by u-bit string, and $M = 2^m$. For any binary matrix A with m rows and u columns:

$$h_A(x) = Ax \pmod 2$$
$$\text{Claim: } \{h_A : A \in \{0,1\}^{m \times u}\} \text{ is universal.}$$

H can be used for dictionaries. In addition to storing the hash table, matrix A also needs to be stored

- Additional storage overhead $\theta(logN \cdot logU)$ bits, if $M = \theta(N)$.

Perfect Hashing

Quadratic Space

If H is *universal* and $M = N^2$, then if h is sampled uniformly from H, the expected number of collisions is < 1. Therefore, there is a hash function $h : U \to [N^2]$ from H for which there are no collisions.

2-Level Scheme

If H is *universal*, then if h is sampled uniformly from H:

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

Amortized Analysis

Amortized analysis a strategy for analyzing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive. We will only look at deterministic algorithms.
An amortized analysis *guarantees* the average performance of each operation in the worst case.

Types of Amortized Analysis

Aggregate method

We count the complexity of each operation and determine a pattern and come up with an overall bound for the time complexity.
Consider a queue with two operations:

1. $INSERT(x)$: insert an element x
2. $EMPTY()$: deletes all elements one by one

Worst case cost of a single $INSERT(x)$: $\theta(1)$
Worst case cost of a single $EMPTY()$: $\theta(n)$
Observe that if there are k $INSERT$, then the sum of cost of all $EMPTY$ is $\leq k$. Total cost: $\leq k + k = 2k \leq 2n$.
Amortized cost is $O(1)$.

Accounting method

Charge i-th operation a fictitious **amortized cost** $c(i)$. This amortized cost $c(i)$ is a fixed cost for each operation, while the true cost $t(i)$ varies depending on what operation is called.
Amortized cost $c(i)$ must satisfy, for all n:

$$\sum_{i=1}^{n} t(i) \leq \sum_{i=1}^{n} c(i)$$

The total amortized costs provides an upper bound on the total true costs. Different operations can have different amortized costs.

Potential method

ϕ: Potential function associated with the algorithm/DS
$\phi(i)$: Potential at the end of the ith operation
Important conditions to be fulfilled by ϕ

- $\phi(i) \geq 0$ for all i

Amortized cost of i-th operation
= Actual cost of ith operation + ($\Delta\phi(i)$)
= Cost of ith operation + ($\phi(i) - \phi(i-1)$)

Amortized cost of n operations
$=\Sigma_i$Amortized cost of ith operation
=Actual cost of n operations + ($\phi(n) - \phi(0)$)
$\geq$ Actual cost of n operations $-\phi(0)$

Select a suitable potential function ϕ, so that for the costly operation, $\Delta\phi_i$ is negative such that is nullifies or reduces the effect of the actual cost.
Try to find a quantity that is decreasing in the expensive operation