1 Math Fundamentals

Formulae

Asymptotics Let f and g be non-negative functions, then $f(n)$ is...

	...if and only if...	if $\lim_{n\to\infty} \frac{f(n)}{g(n)}$
$O(g(n))$	$\exists c, N : f(n) \leq c \cdot g(n)$ for $n \geq N$	$\neq \infty$.
$o(g(n))$	$\forall c, \exists N : f(n) \leq c \cdot g(n)$ for $n > N$	$= 0$.
$\Omega(g(n))$	$\exists c, N : f(n) \geq c \cdot g(n)$ for $n \geq N$	$\neq 0$
$\omega(g(n))$	$\forall c, \exists N : f(n) \geq c \cdot g(n)$ for $n > N$	$= \infty$.
$\Theta(g(n))$	$f(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$	$\neq 0, \infty$

Implications	Equivalences
$f = o(g) \Rightarrow f = O(g)$	$f = O(g) \Leftrightarrow g = \Omega(f)$
$f = \omega(g) \Rightarrow f = \Omega(g)$	$f = o(g) \Leftrightarrow g = \omega(f)$
	$f = O(g)$ and $f = \Omega(g) \Leftrightarrow f = \Theta(g)$

Generalizations: For all $a, b > 0$, $k > 1$, and $n \geq 1$

$$\begin{cases} (\log n)^b = o(n^a) & \\ n^b = o((1+a)^n) & \\ a^{\sqrt{\log n}} = o(n^b) & \\ k^n > n! > n^{n^b} > n^a > \log n > n^{1/k} > O(1) & \approx \text{comparisons} \\ n! \approx n^n \cdot e^{-n} \cdot \sqrt{2\pi n} & \text{Stirling's Formula} \end{cases}$$

Master Theorem: Let $T(n) = aT(n/b) + cn^k$ for $a \geq 1, b \geq 2, c, k \geq 0$. Then $T(n)$ is

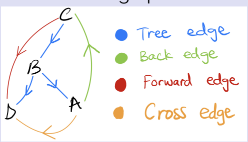
$$\begin{cases} \Theta(n^k) & \text{if } a < b^k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

2 Graph Algorithms

Depth First Search Algorithm

DFS Algorithm **Runtime:** $O(|V| + |E|)$

1. Goes as deep as possible in the graph then backtrack.
2. (In-class algorithm) marks preorder and postorder numbers of each node.
3. In a directed graphs, DFS will label edges as follows. No cross edges in undirected graphs.



Applications:

Detecting cycles	$\exists$ (u, v): postorder(u) < postorder(v)
Topological Sort on **DAGs**	Decreasing post-order
Strongly Connected Components	Hint: DFS on reversed Graph.

Single Source Shortest Paths

BFS Algorithm **only for unweighted graphs.**

1. Goes level by level in the graph.
2. Uses a queue to process nodes.

Runtime: $O(|V| + |E|)$

Let **update(u, v)** be defined as: if $Dist[u]$ + length$(u, v) < Dist[v]$, update: $Prev[v] = u$ and $Dist[v] = Dist[u]$ + length(u, v).

Dijkstra's Algorithm: **only +ve weighted graphs**

1. Let $Dist[v] = \infty$ and $Prev[v] =$ NIL for all vertices v.
2. Let $Dist[s] = 0$ and initialize MinHeap with $(s, 0)$.
3. While heap isn't empty, keep poping the vertex (say u) with the smallest distance. For each neighbor v of u with weight w, **update(u, v)**.

Runtime: $O(|V| * popMin + |E| * Insert)$

Bellman-Ford Algorithm: **Only + and −ve weighted graphs**

1. Let $Dist[v] = \infty$ and $Prev[v] =$ NIL for all vertices v.
2. Let $Dist[s] = 0$.
3. For $|V| - 1$ times: For each edge (u, v), **update(u, v)**
4. Checking for negative weight cycles: repeat the above step once. If any distance can still be improved, a negative weight cycle exists. Hence, return **Inconclusive**

Runtime: $O(|V| * |E|)$

Linear Runtime: **only for Directed Acyclic Graphs (DAGs)**

1. Run DFS to get topological sort.
2. For every edge (u, v), in topological sort, **update(u, v)**.

Runtime: $O(|V| + |E|)$

Min Heaps

Representation: Visually a complete tree. Implementationwise, let $A[0..n-1]$ be a list where $A[0]$ is the root and for any ith element, its parent, left, and right children are at $\lfloor i/2 \rfloor, 2i, 2i+1$ respectively.

Heap Property: The parent element is smaller than its children.

Operations	Description
$Insert(a)$	let $A[n] = a$ and $HeapifyUp(n)$
$PopMin$	let $A[0] = A[n-1]$ and $HeapifyDown(0)$
$HeapifyUp(i)$	repeatedly swap $A[i]$ with its parent until the heap property is restored
$HeapifyDown$	repeatedly swap $A[i]$ with its smallest child until the heap property is restored

Heaps Operations and Runtimes: Both $O(\log n)$ with binary heaps.

Note: Can do better with Fibo-Heaps (amortized $O(1)$ for PopMin.)

Minimum Spanning Trees

Basic Properties:

1. **a Tree** is connected, acyclic, and has $|V| - 1$ edges (any two implies the third).
2. **Cut Property** states that for any cut of a connected, undirected graph, the minimum weight edge that crosses the cut belongs to the MST.
3. Only for connected, undirected, and weighted (non-negative) graphs.

Prim's Algorithm:

1. Start with a single vertex and greedily add closest vertices.
2. Similar to Dijkstra's algorithm, but $dist[v]$ is the weight of the edge connecting v to the MST instead of the distance from s.

Runtime: $O(|E| \log |V|)$ with Fibo-heaps

Kruskal's Algorithm:

1. Sort edges in ascending order of weight.
2. Repeatedly add the lightest edge that does not create a cycle until we have $|V| - 1$ edges.

Runtime: nT(Union) $+ mT$(Find) $+ T$(Sort m Edges).

Notes: Implemented using a union-find data structure.

Disjoint Forest Data Structure

Maintain disjoint sets that can be combined ("unioned") efficiently. Operations MakeSet(x), Find(x), and Union(a, b)

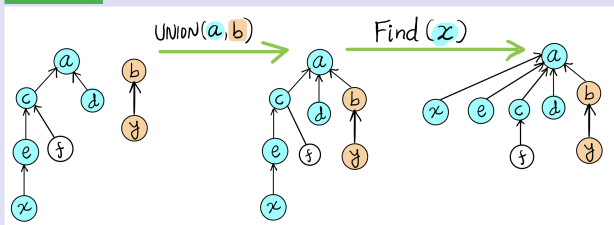
Runtime: Any sequence of m UNIONs and n FINDs operations take $O((m+n)\log^* n)$.

Note: $\log^* n$ is the number of times we can $\log_2 n$ until we get ≤ 1.

Heuristics:

- Union by rank. When performing a union operation, we prefer to merge the shallower tree into the deeper tree.
- Path compression. After performing a find operation, attach all the nodes touched directly onto the root of the tree.

Example:



3 Greedy Algorithms

Main idea: At each step, make a locally optimal choice in hope of reaching the globally optimal solution.

Horn Formula

Algorithm: Set all variables to false and greedily set ones to be true when forced to.

Runtime: linear time in the length of the formula (i.e., the total number of appearances of all literals).

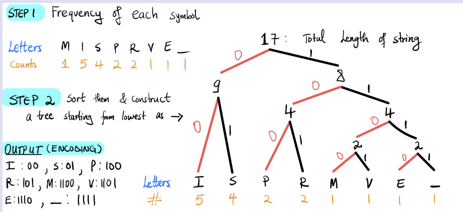
Notes: Only works for SAT instances where in each clause, there is at most one positive literal.

Huffman Coding

Algorithm: **Runtime:** $O(n \log n)$

Find the best encoding by greedily combining the two least frequently items. Optimal in terms of encoding one character at a time.

Example: A Huffman tree for string "Mississippi River"



Set Cover

Given $X = \{x_1, \ldots, x_n\}$, and a collection of subsets $\mathcal{S}$ of X such that $\bigcup_{S \in \mathcal{S}} S = X$, find the subcollection $\mathcal{T} \subseteq \mathcal{S}$ such that the sets of $\mathcal{T}$ cover X.

Algorithm: **Runtime:** $O(|U|)$

1. Greedily choose the set that covers the most number of the remaining uncovered elements at the given iteration.

claim: Let k be the size of the smallest set cover for the instance $(X, \mathcal{S})$. Then the greedy heuristic finds a set cover of size at most $k \ln n$.

Note:

Not always optimal; achieves $O(\log n)$ approximation ratio.

4 Divide and Combine

Main Idea: Divide the problem into smaller pieces, recursively solve those, and then combine their results to get the final result.

Famous Examples w/ Runtimes

Mergesort	$O(n \log n)$
Min and Max on a line	$\frac{3}{2}n - 2$ comparisions; $O(n)$ runtime.
Closest Pair of Points	$O(n \log^2 n)$

n-digit Integer Multiplication

standard Multiplication	$\Theta(n^2)$
3 products on $n/2$ digits	$\Theta(n^{\log_2 3}) = \Theta(n^{1.59})$
5 products on $n/3$ digits	$\Theta(n^{\log_3 5}) = \Theta(n^{1.46})$

$n \times n$ Matrix Multiplication

Strassen's Algorithm: **Runtime:** $O(n^{\log_2 7})$

Divide into four submatrices, each of size $n/2$ by $n/2$.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Find: $P_1 = A(F - H)$, $P_2 = (A + B)H$, $P_3 = (C + D)E$, $P_4 = D(G - E)$, $P_5 = (A + D)(E + H)$, $P_6 = (B - D)(G + H)$, $P_7 = (C - A)(E + F)$, then:

$AE + BG = -P_2 + P_4 + P_5 + P_6$ and $AF + BH = P_1 + P_2$
$CE + DG = P_3 + P_4$ and $CF + DH = P_1 - P_3 + P_5 + P_7$

5 Dynamic Programming

Main Idea: Maintain a lookup table of correct solutions to sub-problems and build up this table towards the actual solution.

Steps:

1. Define subproblems and recurrence to solve subproblems.
2. Combine with **reuse**.
3. Runtime and space analysis.

Edit Distance

Find the minimum number of operations required to transform one string, $A[1 \ldots n]$, into another, $B[1 \ldots m]$.

Algorithm: **Runtime and Space:** $O(nm)$

1. Subproblem: let $D(i, j)$ represent the edit distance between $A[1 \ldots i]$ and $B[1 \ldots j]$.
2. Recurrence is:
   Base cases: $D(i, 0) = i$, $D(0, j) = j$.
   $D(i, j) = \min[D(i - 1, j) + 1, D(i, j - 1) + 1, D(i - 1, j - 1) + (1$ if $i = j$, 0 otherwise$)]$.
3. return $D(n, m)$.

All Pairs Shortest Paths

Given a graph G with n vertices and m edges, calculate distances of the shortest paths between *every* pair of nodes.

Floyd-Warshall Algorithm: **Runtime:** $O(n^3)$

1. Subproblem: let $D_k[i, j]$ represent the shortest path between i and j using only nodes in $[1 \ldots k]$.
2. Recurrence is:
   $$\begin{cases} D_0[i, j] = d_{ij} \text{ if } i \text{ and } j \text{ are connected, } \infty \text{ otherwise.} \\ D_k[i, j] = \min(D_{k-1}[i, j], D_{k-1}(i, k) + D_{k-1}[k, j]). \end{cases}$$
3. return $D(i, j, n)$.

Notes: Does not work for cyclic graphs.

Hashing and Set Resemblance

Primality Testing

Algorithm: Generate large (d-digit) primes: Generate a random d-digit number, Check if it is prime, If not, repeat.

Facts 1: k^{th} prime number is $\Theta(k \log k)$.
Fact 2: of the integers $1, \ldots, n$, $\Theta(n/\log n)$ are prime.
How many d-digits generations until prime: $O(d)$.

Fermat's Little Theorem:
If p is prime and a is not divisible by p, then $a \in \mathbb{Z}$, $a^{p-1} \equiv 1 \pmod p$.
Notes: a-**Pseudoprime** p: $a^{p-1} \equiv 1 \pmod p$, but p is not prime.

Carmichael number: composite number n s.t $a^{n-1} \equiv 1 \pmod n$ for all $a \in \mathbb{Z}$. Example: 561

Miller-Rabin Primality Test:
If p is prime, the only solutions to $a^2 \equiv 1 \bmod p$ are $a \in \{\pm 1\}$.
A **non-trivial square root of 1** is an integer a such that $a^2 \equiv 1 \pmod n$ and $a \not\equiv \pm 1 \pmod n$.

The Test

1. Choose a random $a \in [n]$
2. if $a^{n-1} \not\equiv 1 \pmod n$, return n is composite and a is a witness.
3. Let $n - 1 = 2^t u$ and compute $a^u, a^{2u}, \ldots, a^{2^t u}$.
4. Check for $a^{2^{i-1}u} \not\equiv \pm 1 \pmod n$, $a^{2^i u} \equiv 1 \pmod n$
5. If so, we've found a *non-trivial square root* of 1 modulo n.

a is a **witness** to the compositeness of n, if n fails the test under a.
Note: Let $\mathbf{F}$ be incorrectly identifying n as prime. $\mathbf{Pr(F)} \le \mathbf{1/4}$. With k independent tests, $\mathbf{Pr(F)} \le \mathbf{4^{-k}}$.

Euclid's algorithm

Def: $gcd(a, b) = max(d \in \mathbb{Z} : d|a$ and $d|b)$.
Basic algorithm: find the $gcd(a, b)$ by repeatedly subtracting the smaller number from the larger number.

Extended Euclid's algorithm: find x, y such that $ax + by = gcd(a, b)$.

```
def gcd(a, b):        # a >= b >= 0
    if b == 0:
        return a
    return gcd(b, a mod b)

def Extended-Euclid(a,b):        # a >= b >= 0
    if b==0: return (a,1,0)
    Compute k such that a = bk + (a mod b)
    (d,x,y) = Extended-Euclid(b, a mod b)
    return (d,y,x-ky)
```

Example: Run extended algorithm on $a = 35, b = 15$.

- $gcd(35, 15) = gcd(15, 5) = gcd(5, 0) = 5$.
- $35 = 2 \cdot 15 + 5$.
- $15 = 3 \cdot 5 + 0$.
- $5 = 1 \cdot 0 + 5$.
- $x = 1, y = -2$.

Cryptography

Receiver: picks two large prime numbers p and q, compute $n = pq$.

- public key: (n, e) where e is random s.t. $gcd(e, (p-1)(q-1)) = 1$.
- private key: $\mathbf{d} = e^{-1} \pmod{(p-1)(q-1)} \implies de \equiv 1 \pmod{(p-1)(q-1)}$.

RSA algorithm: Let sender's message be m.

1. Encrytpion: $y = E(m) = m^e \pmod n$ —using public key (n, e)
2. Decryption: $m = D(y) = y^d \pmod n$ —using private key d

NP-c approximations

α**-approximation:** f is an α-approximation to f^* if $f(x) \le \alpha f^*(x)$ for all x.

Independent Set: Given a graph $G = (V, E)$, find largest set of vertices $S \subseteq V$ such that no two vertices in S are adjacent.

Vertext Cover: Given a graph $G = (V, E)$, find the smallest set of vertices $S \subseteq V$ such that every edge in E is incident to at least one vertex in S.

Max Cut: Given a graph, divide vertices into two sets to maximize number of edges between them.

- Split vertices arbitrarily.
- While moving a vertex impoves the solution, move it.
- stop when no more moves improve the solution.

NP-c Heuristics

MAX SAT: Linear Relaxation

- Convert formula to integer equations. E.g. $(X \vee \bar{Y}) \to x' + (1 - y') \ge 1$.
- Relax the constraint for variables $x', y', \cdots \in \{0, 1\}$ to $x', y', \cdots \in [0, 1]$.
- Solve the relaxed problem to get assignment $x', y', \ldots$ Then let $X = 1$ with probability x'

Claim: If formular is satisfiable, then the number of clauses satisfied by above algorithm $\ge$ |clauses| $\cdot (1 - 1/e) \approx 0.63$ |clauses|

MAX SAT: Local Search

- Pick a random assignment $x, y, \cdots \in \{0, 1\}$.
- While moving a variable improves the solution, move it.
- stop when no more moves improve the solution.

Linear Programming

Simplex algorithm: Hill climbing on feasible region always yields an optimal solution.

Network Flow

Story: Given the number of available tickets t_{ij} between cities i and j, along with the city map, the goal is to maximize the number of tickets sold for people traveling from city s to city t.

Reduction to LP: **Objective function:**
Maximize the total flow of tickets from the source node s to the target node t. In other words, maximize the sum of x_{st} over all edges (s,t).

$$\text{maximize} \sum_{(s,t)} x_{st} \tag{1}$$

Subject to the following constraints:
1. Flow conservation constraints: For each node i, other than s and t, the flow into the node must equal the flow out of the node.

$$\sum_{j} x_{ij} - \sum_{k} x_{ki} = 0, \forall i \neq s, t \tag{2}$$

2. Capacity constraints: The flow of tickets on each edge (i, j) must not exceed the available tickets t_{ij}.

$$0 \leq x_{ij} \leq t_{ij}, \forall (i, j) \tag{3}$$

Ford Fulkerson:

- Make a Residual graph and Start with empty flow.
- While there is an augmenting path from s to t:
  - Find the bottleneck capacity b on the path.
  - Increase the flow on the path by b and update residues.
- The final flow is the maximum flow.

Augmenting path: is a path from s to t such that the flow on the path can be increased by at least one unit.
Residual Graph: for every edge (x,y) of G, add edge (y,x), capacity c(y,x) = flow f(x,y)

Runtime:

- with DFS to find augmenting paths$O(VEU)$, where U is max edge capacity.
- $O(VE^2)$ with BFS ("Edmonds-Karp algorithm").

Max-flow min-cut theorem: The maximum flow is equal to the minimum capacity of an s-t cut.

Claim: when* Ford-Fulkerson terminates, we can find a cut matching the flow.

- No s-t path (of nonzero edges) in residual graph.
- Choose V1 = vertices reachable from s
- All edges e leaving V1 have f(e) = c(e)
- All edges e entering V1 have f(e) = 0
- So, total flow = capacity of cut.