1 Math Fundementals

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 $n \geq 1$

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

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

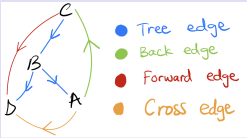
$$\begin{cases} \Theta(n^k) & \text{if } a < b^k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

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 for every node.
3. In a directed graphs, DFS will label edges into (tree, forward, back, cross). No cross edges in undirected graphs.



Applications:

Detecting cycles	Check for a back edge
Topological Sort on **DAGs**	Descending post-order
Strongly Connected Components	Hint: DFS on reversed Graph.

Single Source Shortest Paths

BFS Algorithm **SSSP for unweighted graphs only.**

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

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

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 $(s, 0)$.
2. Keep poping the vertex with the smallest distance.
3. For each neighbor of the popped vertex, update the distance if it is smaller and add it to the heap. Repeat 2. until heap is empty.

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

Bellman-Ford Algorithm: **Both +/−ve weighted graphs**

1. Let $Dist[v] = \infty$ and $Prev[v] =$ NIL for all vertices v.
2. Let $Dist[s] = 0$.
3. Repeat the following $|V| - 1$ times:
   - For each edge (u, v) with weight w, if $Dist[u] + w < Dist[v]$, update $Dist[v] = Dist[u] + w$ and $Prev[v] = u$.

4. Checking for negative weight cycles: repeat the above step once. If any distance can still be improved, a negative weight cycle exists.

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

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

1. Run DFS to get topological sort.
2. Visit all edges coming out of nodes in the topological order. If multiple edges coming out, pick the shortest.

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

1. Start with a single vertex and add edges greedily.
2. Implemented in a similar fashion as Dijkstra's, where we add vertex v to the MST when we pop it from the heap.

Kruskal's Algorithm: **Runtime:** $O(|E| \log^* |V|)$ amortized

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.

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)

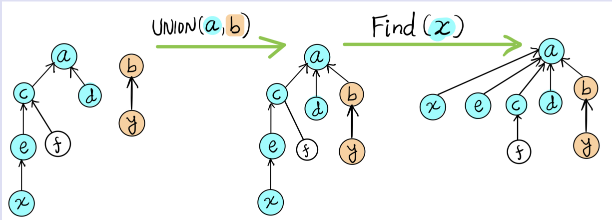
Runtime: m UNION $+n$ FIND operations take $O((m+n)\log^* n)$.

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).

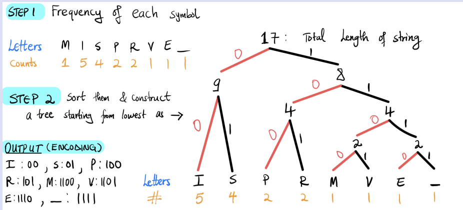
Notes: Only works for SAT instances where in each clause, there is at most one positive literal.

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 $X = \{x_1, \dots, x_n\}$, 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, $A[1 \dots n]$, into another, $B[1 \dots m]$.

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

1. Subproblem: let $D(i, j)$ represent the edit distance between $A[1 \dots i]$ and $B[1 \dots j]$.
2. Recurrence is:
   Base cases: $D(i, 0) = i, D(0, j) = j$.
   $D(i, j) = \min[D(i-1, j) + 1, D(i, j-1) + 1, D(i-1, j-1) + (1 \text{ if } i = j, 0 \text{ otherwise})]$.
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 $[1 \dots k]$.
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.

(in-class) DP Examples

- String Reconstruction(Word Break
- LongestIncrSubsequence (LIS)
- Number of LISs
- Context-free grammar parsing $O(n^3 m^2)$
- Knapsack Problem $O(n \cdot |capacity|)$ NP
- DP on trees: Dominating Set
- Traveling salesman problem