Competitive Programming Notes

Fatima Afzali, Fall/Spring 2019

Topics

Dynamic Programming

- ~~Bellman's Equation~~
- ~~Longest Increasing Subsequence~~
- ~~Longest Common Subsequence~~
- ~~Subset Sum~~
- ~~0-1 Knapsack~~
- ~~Levenshtein Distance~~

Trees

- Heaps
- Binary Search Trees
- Red-Black Trees
- AVL Trees
- Segment Trees
- Minimum Spanning Trees (Prim, Kruskal)
- Tries

Graphs

- Shortest Paths (~~Dijkstra~~, Bellman-Ford, Floyd-Warshall, BFS, DFS)
- Eulerian/Hamiltonian Paths
- (Strongly) Connected Components
- Cut Vertices
- Transitive Closures
- Bipartite Graphs
- Topological Sorting / DAGs
- Clique Finding
- König's Theorem
- Maximum Matching
- Network Flow (Dinic, Ford-Fulkerson, Edmonds-Karp)

Math Stuff

- ~~Calculating Combinations~~
- ~~Derangements~~
- ~~Inclusion-Exclusion Principle~~
- ~~Euler's Totient Function~~
- ~~GCDs/LCMs~~
- ~~$\mathbb{Z}/p\mathbb{Z}$, Modular Arithmetic~~
- ~~Chinese Remainder Theorem~~
- Hensel's Lifting Lemma
- Matrix Calculations
- Linear Programming
- Convex Optimization

Computational Geometry

- Line Angles, Intersections
- Convex Hull
- Voronoi Diagrams
- Delaunay Triangulation
- K-D Trees

1 Dynamic Programming

1.1 Bellman's Equation

Suppose that a decision process has several different states $X = \{x_i\}$, that for each state there is a set of actions $\Gamma(x) = \{a_i\}$, each of which takes us to another state $x' = T(x,a) \in X$, and that moving from a state x to another state via an action $a \in \Gamma(x)$ yields a payoff of $F(x,a)$. The set of states and actions can be organized into a graph; suppose this graph is finite, directed, and acyclic, so that in particular we can represent X as $X_0 \cup X_2 \cup \ldots \cup X_N$, where each member of X_i has an arrow to X_j only if $j > i$.

Let $V(x_i)$ be the highest payoff obtainable by any path $x_i \xrightarrow{a_i} x_{i+1} \xrightarrow{a_{i+1}} \cdots \to x_N$. Then, Bellman's equation states that

$$V(x) = \max_{a \in \Gamma(x)} F(x,a) + V(T(x,a))$$

1.2 Example: Longest Increasing Subsequence

As an example, suppose we are given an array $S = [s_0, \ldots, s_{N-1}]$, and we want to decide the longest increasing subsequence $S' = [s_{i_0}, s_{i_1}, \ldots, s_{i_n}]$ of S (i.e., $s_{i_j} < s_{i_k} \iff i_j < i_k \iff j < k$). This is representable as a decision process where a stage $x \in X_i$ is a pair (B,i), where B is a sequence of booleans of length N corresponding to an increasing subsequence of S. (The i is just so we can differentiate two identical arrays at different stages). Any state (B,i) has two descendants, one in which $B[N-1-i]$ is set to true, and one in which $B[N-1-i]$ is set to false. The payoff for turning a bit on is 1, and for leaving a bit off is 0. Letting $x_0 = (S,0)$, $V((S,0))$ is the maximum length of any increasing subsequence of S, so we have

$$V((S,0)) = \max(1 + V(\text{we pick } S[N-1]), V(\text{we don't pick } S[N-1]))$$

Since this is recursive, we have to start from the bottom and work our way up; we end up with an $O(N^2)$ algorithm.

1.3 Optimal Substructure

Formally, we can use dynamic programming to solve an problem whenever we can manipulate the problem into a way that allows us to apply the Bellman equation. Qualitatively, this requires self-similarity: a problem can be broken up into smaller problems that look just like it, a quality known

as optimal substructure; LIS is an example of this. Often, we can just wave our hands around and replace the Bellman equation with a recurrence relation.

1.4 Example: Longest Common Subsequence

Given two arrays S, T of length m, n, what's the length $LCS(S,T)$ of the longest subsequence common to both S and T? Recursively, it's either $LCS(S[0,\ldots,m-2],T[0,\ldots,n-2])+1$ in the case where $S[m-1]=T[n-1]$, or it's $\max(LCS(S,T[0,\ldots,n-2]),LCS(S[0,\ldots,m-2],T))$. In the base case, $LCS(\varnothing,T[0,\ldots,i])=LCS(S[0,\ldots,j],\varnothing)=0$. The naive recursive implementation has us calculate the same value several times over (q.v. calculating the Fibonacci sequence), so we may use an $m\times n$ matrix to store subsolutions, working our way through it in $O(mn)$ time to get to the end, which will contain $LCS(S,T)$. This is a common theme of dynamic programming: the organization of recursively dependent values into a matrix, and iteration through that matrix in order to find the final answer.

1.5 Example: Subset Sum

Given an array S of n non-negative integers and an integer $k>0$, is there a subset of S that adds up to k? Recursively, this is true if either there's a subset of $S[0,\ldots,n-2]$ that adds up to k, or a subset of $S[0,\ldots,n-2]$ that adds up to $k-S[n-1]$. Again, we sidestep the inefficiency of the naive recursive solution by using an $n\times(k+1)$ matrix M of booleans where $M[i,j]$ is True iff $S[0,\ldots,i-1]$ has a subset that adds up to j. Our recursion relation set translates to $M[i,j]=M[i-1,j]\ ||\ M[i-1,j-S[i]]$, with $M[0,j]=(j\in S)$, and the answer is $M[n-1,k]$, found in $O(kn)$ time.

1.6 Example: 0-1 Knapsack

Given a set of items indexed as $0,\ldots,n-1$, each with a positive value $V[i]$ and positive weight $W[i]$, as well as a maximum weight w, what's the maximum value $K(V,W,w)$ of any set $[i_0,\ldots,i_s]$ that satisfies $\sum_j W[i_j]\leq w$? Recursively, it's either $V[n-1]+K(V[0,\ldots,n-2],W[0,\ldots,n-2],w-W[n-1])$ in the case that we should pick $W[i]$, or it's $K(V[0,\ldots,n-2],W[0,\ldots,n-2],w)$ in the case that we shouldn't. Let M be an $(n+1)\times(w+1)$ matrix where $M[i,j]=K(V[0,\ldots,i-1],W[0,\ldots,i-1],j)$, s.t. our recurrence relation translates to $M[i,j]=\max(M[i-1,j],V[i-1]+M[i-1,j-W[i-1]])$. The first row is vacuously 0, giving us our base case, as well as is our first column. Iterating through the matrix and returning $M[n,w]$ yields the answer in $O(nw)$ time.

1.7 Example: Levenshtein Distance

Suppose we formed a graph out of the set of all strings by connecting those strings separated by a single insertion/deletion (cat $\leftrightarrow$ at, at $\leftrightarrow$ bat) or by a single substitution (cat $\leftrightarrow$ bat). The Levenshtein distance between two strings S, T is their distance $L(S,T)$ in this graph; how can we efficiently calculate it? Letting $|S| = m$ and $|T| = n$, we note that $L(S,T)$ is either $L(S[0,\ldots,m-2],T)+1$ (if we have to add to S), $L(S,T[0,\ldots,n-2])+1$ (if we have to add to T), $L(S[0,\ldots,m-2],T[0,\ldots,n-2])+1$ (if we have to substitute), or $L(S[0,\ldots,m-2],T[0,\ldots,n-2])$ (if we don't have to substitute). Since we can add and delete anywhere we want, the position of the strings relative to one another is immaterial. Letting M be an $(m+1)\times(n+1)$ matrix where $M[i,j] = L(S[0,\ldots,i-1],T[0,\ldots,j-1])$, we have our recurrence relation: $M[i,j] = \min\big(M[i-1,j]+1, M[i,j-1]+1, M[i-1,j-1]+(1-\delta_{S[i-1],T[j-1]})\big)$. When one prefix is zero, the Levenshtein distance between the two prefixes is the length of the other, so $M[0,j] = j$ and $M[i,0] = i$, giving us our base case. Iterating through the matrix and returning $M[m,n]$ returns $L(S,T)$ in $O(mn)$ time.

2 Trees

A tree is a connected graph with no cycles. Equivalent conditions for a finite graph G to be a tree:

1. Adding any edge would form a simple cycle.
2. Removing any edge would disconnect the graph.
3. A unique path connects any two vertices.
4. The graph is connected and has $|V(G)|-1$ edges.

There are n^{n-2} trees on n labeled vertices (Cayley's theorem).

Algorithmically, trees are generally considered to start at a root, with each child considered the root of a subtree; data is also attached to the nodes, generally in the form of integers. Each node has only local knowledge – what its parents and children are, and what integer it contains – making tree traversal primarily recursive and making tree surgery simple. From this point of view, the height of a node is the length of its (unique) path to the root.

2.1 Binary Heaps

A binary heap is a binary tree that is complete as it can be: if the highest node has height h from the root, then there are 2^k nodes of height $0 \le k < h$, and every node with a complete right subtree

has a complete left subtree. We can implement a binary heap as an array, reading nodes from top to bottom and left to right. Using 1-based indexing, the parent of node i is then $\lfloor i/2 \rfloor$, the left child is $2i$, and the right child is $2i+1$. (In 0-based indexing, these become $\lfloor (i-1)/2 \rfloor$, $2i+1$, and $2i+2$).

Heaps must also satisfy a heap property: for max-heaps, the value of a parent must be at least that of its children, and for min-heaps, the value of a parent must be at *most* that of its children.

NOT DONE

3 Graphs

3.1 Shortest Paths

Given a weighted graph G, and two vertices $v, w \in V(G)$, how do we find the length $D(v,w)$ of the shortest path between v and w?

Dijkstra's Algorithm Given a vertex v, let's find $D(v,w)$ for all $w \in V(G)$. There's an obvious recurrence relation given by $D(v,w) = \min_{u \in V(G)} D(v,u) + D(u,w)$, but we can only take advantage of this if we know how to identify the minimum u when we see it. To facilitate this, create a set of vertices N representing the vertices whose distance from v we know for sure. At first, only $v \in N$, but we also know that the path to the closest neighbor u of v must be (u,v), so add that neighbor to N and set $D(v,u) = W(v,u)$. Intuitively, we can always treat N as a single node, where $w \in V(G)$ is a neighbor of N if it's a neighbor of some $n \in N$, and we can therefore keep going in this manner, finding the neighbor w of N that minimizes $D(v,n) + W(n,w)$, adding it to N, and setting $D(v,w) = D(v,n) + W(n,w)$.

NOT DONE

4 Number Theory

4.1 Integers

Algebraically, $\mathbb{Z}$ is a commutative ring: $+$ defines an abelian group with identity 0, and $\cdot$ is associative, commutative, and distributes over $+$. Being a Euclidean domain, we see that (a) all ideals are principal, (b) the prime elements are the irreducible elements are the prime numbers (and their additive inverses), (c) greatest common divisors exist and are linear combinations of their arguments, and (d) every element has a unique decomposition as a product of primes.

Greatest Common Divisors The greatest common divisor of two elements a, b of a ring is, when it exists, any c such that $a, b \in (c)$ and, for any d, $a, b \in (d) \implies c \in (d)$. In $\mathbb{Z}$, we can calculate this c using two observations: when $b \neq 0$, $\gcd(a, b) = \gcd(b, a \bmod b)$, and when $b = 0$, $\gcd(a, b) = a$. This leads to the Euclidean algorithm. Since $a \bmod b$ is the remainder of Euclidean division by b, we can extend the Euclidean algorithm, keeping track of the integer quotients, in order to figure out how to express $\gcd(a, b)$ as a linear combination of a and b. Dually, the least common multiple of a, b is any c such that $c \in (a), (b)$ and for any d, $d \in (a), (b) \implies d \in (c)$; in $\mathbb{Z}$, this is $ab/\gcd(a, b)$.

Primality By the FTA, a number is prime n if and only if there is no prime $1 < p < n$ with $p \mid n$. Supposing that n is not a square, any prime $p > \sqrt{n}$ that divides n must yield some prime $q \mid n/p \mid n$ with $q < \sqrt{n}$; as such, n is prime if and only if there is no prime $1 < p \leq \sqrt{n}$ with $p \mid n$. This yields an $O(n^{1/2})$ primality testing algorithm known as trial division. Since it is constructive, giving an explicit factor of n whenever n isn't prime, it gives us an algorithm for finding the prime factors of n.

The failure of an $n > 1$ to be prime is given by the number of integers $1 \leq m \leq n$ not coprime to n, with $\gcd(m, n) \neq 1$. We denote the number of integers that *are* coprime to n by $\varphi(n)$, the corresponding function being known as Euler's totient function.

4.2 Modular Arithmetic

For $n \in \mathbb{Z}^+$, we can quotient $\mathbb{Z}$ by the ideal $(n) = n\mathbb{Z}$ in order to obtain the quotient ring $\mathbb{Z}/n\mathbb{Z}$. If we want to make this a group under multiplication, we must remove all the classes $[m]$ with m not coprime to n, obtaining the group $(\mathbb{Z}/n\mathbb{Z})^\times$. The order of this group is, by definition, $\varphi(n)$; for any g in a group G we have $g^{|G|} = e$, which implies that $a^{\varphi(n)} \equiv 1 \bmod n$ (Euler's theorem). When n is a prime, every $m < p$ is coprime to p, implying that $\mathbb{Z}/p\mathbb{Z}$ has the structure of a field without any additional modification. As a special case of Euler's theorem, we have Fermat's little theorem: $a^{\varphi(p)} = a^{p-1} \equiv 1 \bmod p$. To find the multiplicative inverse of $m \in \mathbb{Z}/p\mathbb{Z}$, i.e. the n such that $mn \equiv 1 \bmod p$, note that we must have $mn + kp = 1$ for some $k \in \mathbb{Z}$. The extended Euclidean algorithm gives us the m, k such that $mn + kp = \gcd(n, p) = 1$, so it must give us this multiplicative inverse.

Quadratic Residues When p is an odd prime, it is useful to analyze those $n \in \mathbb{Z}/p\mathbb{Z}$ such that $n = m^2$, and we can take a "square root" of n. We call such numbers quadratic residues. Noting that $\left(n^{\frac{p-1}{2}}\right)^2 = n^{p-1} \equiv 1 \bmod p$, we must have $n^{\frac{p-1}{2}} \equiv \pm 1$, and therefore $n^{\frac{p+1}{2}} \equiv \pm n^2$, so n is a quadratic residue if and only if $n^{\frac{p-1}{2}} \equiv 1$. We define the Legendre symbol $\left(\frac{n}{p}\right)$ to be $n^{\frac{p-1}{2}}$, which is 0 when $n = 0$,

1 when n is a quadratic residue, and -1 otherwise. The Law of Quadratic Reciprocity states that, for p, q odd primes,

$$\left(\frac{q}{p}\right)\left(\frac{p}{q}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}$$

Chinese Remainder Theorem Suppose that we have a system of equations $x = a_i \bmod n_i$, for $i = 1, \ldots, k$, that we would like to solve for x. Defining $n = n_1 \cdot \ldots \cdot n_k$, the Chinese Remainder Theorem (CRT) asserts that, whenever the n_i are pairwise coprime, there is an isomorphism of rings $\mathbb{Z}/n\mathbb{Z} \to (\mathbb{Z}/n_1\mathbb{Z}) \times \ldots \times (\mathbb{Z}/n_k\mathbb{Z})$ that sends $x \in \mathbb{Z}/n\mathbb{Z}$ to $(x \bmod n_1, \ldots, x \bmod n_k)$, asserting that there is a unique (modulo n) solution to this system of equations. We can construct it explicitly: since $n_i, n/n_i$ are coprime, we can calculate the inverse of n/n_i in $(\mathbb{Z}/n_i\mathbb{Z})$, denoting it $\tilde{n}_i$. Then, $x = a_1\tilde{n}_1(n/n_1) + \ldots + a_k\tilde{n}_k(n/n_k)$ is a solution.

As a corollary of the CRT, $\varphi(ab) = \varphi(a)\varphi(b)$ whenever a, b are coprime.

5 Combinatorics

There are $\binom{n}{k}$ ways to choose a k-element subset of an n-element set, where the binomial coefficient $\binom{n}{k}$ is defined as $n!/(k!(n-k)!)$; the set of such subsets is in bijection with the set of $(n-k)$-element subsets (take complements), so $\binom{n}{k} = \binom{n}{n-k}$. Since there is a unique 0-element set, $\varnothing$, $\binom{n}{0} = \binom{n}{n} = 1$. We have a recurrence relation

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

which allows us to tabulate all values of $\binom{a}{b}$ for $a, b \leq n$ in $O(n^2)$ time and then look up binomial coefficients in $O(1)$ time, as opposed to calculating them in $O(n)$ time whenever we need one.

Inclusion-Exclusion The union of two sets A, B has cardinality $|A \cup B| = |A| + |B| - |A \cap B|$, the intersection being subtracted in order to avoid double counting. For sets A, B, C,

$$|A \cup B \cup C| = |(A \cup B) \cup C| = |A \cup B| + |C| - |(A \cup B) \cap C| = |A| + |B| + |C| - |A \cap B| - |(A \cup B) \cap C|$$

$$= |A| + |B| + |C| - |A \cap B| - |(A \cap B) \cup (A \cap C)| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

The principle extends inductively, with $|A_1 \cup \ldots \cup A_n|$ being equal to the sum of all cardinalities of sets minus the sum of all cardinalities of pairwise intersections plus the sum of all cardinalities of all intersections of three sets minus ... .

As an example, consider the number of derangements of the list $[1, 2, \ldots, n]$, or permutations $\sigma \in S_n$ with no fixed points. Every non-derangement has at least one fixed point, and we can use

inclusion-exclusion to count $|(\text{permutations that fix 1}) \cup \ldots \cup (\text{permutations that fix } n)|$, to see that the number of derangements is given by $D_n = \sum_{k=0}^{n} (-1)^k n!/k!$; this turns out to be equivalent to $\left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor$.

6 Linear Algebra

A (real) $m \times n$ matrix is a linear map $A : \mathbb{R}^n \to \mathbb{R}^m$, with the component A_{ij} defined to be $\pi_i(Ae_j)$.

Multiplication and Exponentiation Matrix multiplication is given by function composition:

$$(AB)_{ij} = \pi_i(ABe_j) = \pi_i\left(A\sum_k B_{kj}e_k\right) = \pi_i\left(\sum_{k,\ell} A_{\ell k}B_{kj}e_\ell\right) = \sum_k A_{ik}B_{kj}$$

If $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$, then $AB \in \mathbb{R}^{m\times p}$, and calculation of AB is $O(mnp)$. In particular, when A and B are square $n \times n$ matrices, calculation of AB is $O(n^3)$; the matrix exponential A^k is defined by $A^0 = I_n$ and $A^k = AA^{k-1} = A^{k-1}A$. We can calculate these rapidly as follows:

$$A^k = A^{\sum_{i=0}^{\lfloor \log_2 k \rfloor} c_i 2^i} = \prod_{i=0}^{\lfloor \log_2 k \rfloor} A^{2^i}$$

where $A^{2^i} = \left(A^{2^{i-1}}\right)^2$ is calculated in $O(in^3)$, making this calculation $O\left(\lfloor \log_2 k \rfloor n^3\right)$.

There is another notion of a matrix exponential, given by

$$e^A = \sum_{i=0}^{\infty} \frac{A^i}{i!}$$

(This is guaranteed to converge by the observation that $\det e^A = e^{\mathrm{Tr}A}$).

Determinants and Inverses Considering an $A \in \mathbb{R}^{n\times n}$ as an automorphism of an n-dimensional $\mathbb{R}$-vector space V with n-vector $v_1 \wedge \ldots \wedge v_n$, the determinant of A is the unique scalar d such that $Av_1 \wedge \ldots \wedge Av_n = d \cdot v_1 \ldots v_n$, i.e. the signed volume of the n-parallelotope obtained by applying A to an arbitrary basis of V. The determinant of a 1×1 matrix $[a]$ is just a; a recursive algorithm is given as follows: let $M_{r,c}$ denote the (r,c)th minor of A, obtained by removing the rth row and cth column. Then, for *any* fixed $1 \le r_0, c_0 \le n$,

$$\det A = \sum_{r=0}^{n} (-1)^{r+c_0} A_{r,c_0} \det M_{r,c_0} = \sum_{c=0}^{n} (-1)^{r_0+c} A_{r_0,c} \det M_{r_0,c}$$

The cofactor $C_{r,c}$ of the matrix A is defined to be $(-1)^{r+c} \det M_{r,c}$. The inverse of the matrix A, i.e. the matrix A^{-1} satisfying $A^{-1}A = AA^{-1} = I_n$, is computed by the formula

$$\left(A^{-1}\right)_{ij} = \frac{C_{ji}}{\det A}$$

Properties of Matrices An $n \times n$ matrix A is symmetric when $A^T = A$, antisymmetric when $A^T = -A$, orthogonal when $A^T = A^{-1}$, positive-definite when $x^T Ay$ is an inner product on $\mathbb{R}^n$, diagonal when $A_{ij} \neq 0 \implies i = j$, and sparse when "most" of its elements are 0. The trace of a matrix is given by $\mathrm{Tr} A := \sum_i A_{ii}$.

Orthogonal matrices are (by definition) invertible, and have determinant ± 1. The group of orthogonal $n \times n$ matrices is the group of isometries of $\mathbb{R}^n$, and is called the orthogonal group $\mathrm{O}(n)$; as a subset of $\mathbb{R}^{n^2}$ with the subspace topology, this consists of two connected components, and the one corresponding to $+1$, consisting of the rotations, is called the special orthogonal group $\mathrm{SO}(n)$.

Two square matrices A and B are similar if there's an invertible S such that $S^{-1}AS = B$ (this is an equivalence relation); A is diagonalizable if it is similar to a diagonal matrix D. Then, $A^n = S^{-1}DSS^{-1}DS \ldots S^{-1}DS = S^{-1}D^nS$ can be computed very rapidly.

A matrix is upper/lower triangular if it contains no elements below/above its diagonal.

Eigenstuff An eigenvector of an $n \times n$ matrix A is a non-zero $v \in \mathbb{R}^n$ satisfying $Av = \lambda v$, for some $\lambda \neq 0 \in \mathbb{R}$ known as the associated eigenvalue. This implies that $(A - \lambda I)v = 0$, which is soluble iff $\det(A - \lambda I) = 0$. $\det(A - \lambda I)$ is a degree n polynomial in λ known as the characteristic polynomial $\chi_A(\lambda)$, so there are at most n (linearly independent) eigenvectors of A. The algebraic multiplicity of λ is the multiplicity of λ as a root of χ_A, and the geometric multiplicity of λ is $\dim \ker A - \lambda I$. The space $\ker A - \lambda I$ is known as the eigenspace of A associated with λ.

The trace of A is the sum of its eigenvalues, and the determinant is the product. A is invertible iff all eigenvalues are non-zero, and the eigenvalues of A^{-1} are the inverses of those of A, carrying the same geometric and algebraic multiplicities. A is positive definite/positive semidefinite/negative semidefinite/negative definite iff all eigenvalues are positive/non-negative/non-positive/negative. Orthogonal matrices have eigenvalues ± 1.

Decompositions When can we factor a matrix into a product of matrices with interesting properties?

A QR decomposition of a non-singular $n \times n$ matrix A is a decomposition $A = QR$, where Q is orthogonal and R is upper triangular. Writing A as $[a_1, \ldots, a_n]$, let $[u_1, \ldots, u_n]$ denote the orthogonal

set given by applying the Gram-Schmidt process to $[a_1, \ldots, a_n]$, and let q_i denote the unit vector $u_i/||u_i||$, such that $[q_1, \ldots, q_n]$ is an orthonormal basis of $\mathbb{R}^n$. Letting $Q = [q_1, \ldots, q_n]$, and defining $R_{ij} = \langle e_i, a_j \rangle$ for $i \leq j$ and 0 for $i > j$, we have $(QR)_{ij} = \sum_k Q_{ik} R_{kj} = \sum_k (e_k)_i \langle e_k, a_j \rangle = (a_j)_i = A_{ij}$ and therefore $A = QR$.

Eigenvalues Suppose we want to find the eigenpairs $(\lambda_1, v_1), \ldots, (\lambda_k, v_k)$ of an $n \times n$ matrix A. One way to do this is by the QR algorithm, which decomposes $A_1 := A$ as $Q_1 R_1$, defines $A_2 = R_1 Q_1$, and continues: Q_i and R_i are the QR decomposition of A_i, and A_{i+1} is $R_i Q_i$. Since $R_i Q_i = Q_i^{-1} A_i Q_i$, the A_i all have the same eigenvalues. A_N will converge to an upper triangular matrix with eigenvalues listed on the diagonal, after some iterations, and we will have obtained $A_N = Q_N^{-1} Q_{N-1}^{-1} \ldots Q_1^{-1} A Q_1 \ldots Q_N = (Q_1 \ldots Q_N)^{-1} A (Q_1 \ldots Q_N)$.

NP-Hard Problems

1. SAT: Is a given Boolean expression satisfiable?
2. Subgraph Isomorphism: Is one graph isomorphic to some subgraph of another?
3. 0-1 Integer Programming: Maximize $\boldsymbol{c}^T\boldsymbol{x}$ subject to $(A\boldsymbol{x})_i \leq \boldsymbol{b}_i$ and $\boldsymbol{x}_i \in \{0,1\}$.
4. Clique problem: Find a clique of size k in a graph with n vertices. (Tractable in planar graphs).
5. Set packing: Given a set S, a $U \subseteq \mathcal{P}(S)$, and a $k \in \mathbb{Z}$, find pairwise disjoint $U_1, \ldots, U_k$.
6. Vertex covering: Find a minimal $\{v_i\} \subseteq V(G)$ such that each $e \in E(G)$ ends at some v_i.
7. Set cover: Find the smallest subset of a cover of a set S which still covers S.
8. Traveling Salesman Problem: Find a Hamiltonian cycle of minimal weight in a graph.
9. Hamiltonian paths: Find a Hamiltonian path in a graph (undirected or directed).
10. Chromatic numbers: Can a graph be k-colored?
11. 0-1 Knapsack: Discussed above.
12. Subset sum: Discussed above.
13. Longest common subsequence: Discussed above. (Dually, shortest common supersequence).

Mathematical Identities

Vandermonde's identity, and a special case:

$$\sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}=\binom{m+n}{r} \qquad \sum_{i=0}^{k}\binom{k}{i}^2=\binom{2k}{k}$$

Hockey stick identity:

$$\sum_{i=r}^{n}\binom{i}{r}=\binom{n+1}{r+1}$$

Stirling's formula (heuristic):

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

Lagrange multipliers: to minimize $f(x_1,\ldots,x_n)$ constrained to $g(x_1,\ldots,x_n)=0$, find λ s.t. $\nabla f-\lambda\nabla g=0$.