


Advanced Algorithms — Problem set 1


Whenever you give an algorithm, also argue for its running time and correctness. Always feel free to ask in the forum or send a mail if you're stuck, need a clarification, or suspect a typo.

1. In this exercise, we extend the triangle detection algorithm from the lecture. Let $G = (V, E)$ be an undirected graph on n vertices. Recall that ω denotes the optimal exponent of matrix multiplication. You may use a subroutine that multiplies $t \times t$ matrices in time $O(t^\omega)$. A k -clique is a vertex subset $T \subseteq V$ of size k such that $uv \in E$ for all $u, v \in T$ with $u \neq v$. Give an algorithm that decides whether G contains a k -clique in $O(n^{\omega k/3})$ time, for k divisible by 3. (Hint: Split the potential clique into three equal-sized parts, brute-force over these parts, and reduce to finding triangles.) 

2. Computing the matrix-vector product Av generally takes $O(n^2)$ arithmetic operations, but for some structured matrices, less operations are required. Let n be a power of two, and let $\omega_{n,0}, \dots, \omega_{n,n-1} \in \mathbb{C}$ be the n -th roots of unity, where $\omega_{n,k} = \cos(2\pi k/n) + i \sin(2\pi k/n)$. Consider the matrix

$$D = \begin{pmatrix} \omega_{n,0}^0 & \dots & \omega_{n,0}^{n-1} \\ \vdots & \ddots & \vdots \\ \omega_{n,n-1}^0 & \dots & \omega_{n,n-1}^{n-1} \end{pmatrix}.$$

Given as input v , compute the matrix-vector product Dv with $O(n \log n)$ arithmetic operations. (Hint: Unsurprisingly, the problem can be rephrased as a FFT.)

3. The *odd Goldbach conjecture* is that any odd integer $n \geq 5$ can be written as the sum of three primes. 
 - (a) Using your own implementation of FFT (this will help you understand the algorithm), verify the conjecture for the maximum integer n you can manage. You may import any libraries for handling primes. In Python: After importing `primerange` and `sieve` from `sympy`, the primes between $a, b \in \mathbb{N}$ can be obtained as `primerange(a,b)`.
 - (b) Bonus: For odd $5 \leq n \leq 10^5$, plot how many triples of primes (p_1, p_2, p_3) satisfy $n = p_1 + p_2 + p_3$. It will be useful to use a scatter plot, where you draw a point $(n, f(n))$ for each relevant n , and $f(n)$ is the number of triples.
4. In this exercise, we show that the determinant $\det(A)$ of an $n \times n$ matrix can be computed from traces of matrix powers. This needs some preparation:

- Given an integer $n \geq 1$, a *partition* λ of n is a sorted tuple (n_1, \dots, n_s) of positive integers with $\sum_{i=1}^s n_i = n$. In other words, a partition is a way of writing $n = n_1 + \dots + n_s$ with positive integers $n_1 \leq \dots \leq n_s$, like $10 = 1 + 1 + 3 + 5$ or $10 = 2 + 3 + 5$. Given a partition λ and $1 \leq \ell \leq n$, write λ_ℓ for the number of times that ℓ occurs in λ . For example, in $\lambda = (1, 1, 3, 5)$ we have $\lambda_1 = 2$, $\lambda_3 = 1$, $\lambda_5 = 1$ and $\lambda_\ell = 0$ for all other ℓ .
- For an $n \times n$ matrix M , its *trace* is the sum of diagonal entries $\text{tr}(M) = \sum_{i=1}^n M_{i,i}$.
- The so-called Newton identities show that

$$\det(A) = (-1)^n \sum_{\lambda} \prod_{\ell=1}^n \frac{\text{tr}(A^\ell)^{\lambda_\ell}}{\lambda_\ell! (-\ell)^{\lambda_\ell}}, \quad (1)$$

where λ ranges over all partitions of n . You can use this without proof.

Using the above, show how to compute the determinant in $O(n^{\omega+1})$ operations.¹ (Hint: You don't have enough time to sum over all partitions λ , as their number grows super-polynomially. Instead, use dynamic programming to construct any partition λ in n steps. At step $\ell = 1, \dots, n$, decide how often the number ℓ should occur in λ . In other words, commit to the value of λ_ℓ . This helps, because the factor corresponding to ℓ in (1) depends only upon λ_ℓ .)

¹This number of operations is not too great, because Gaussian elimination would already do the job in $O(n^3)$ operations. (After you brought the matrix into triangular form via Gaussian elimination, the determinant can be read off as the product of diagonal entries.) But with additional tricks, which we won't cover here, the running time can be improved to $O(n^\omega)$. This means computing determinants is asymptotically not harder than multiplying matrices.