Efficient Algorithms and Intractable Problems

Daniel Deng

1 Complexity Analysis

Definition 1.1 (Asymptotic Notations)**.**

$$\begin{aligned}
f = O(g) &\approx f(n) \leq c \cdot g(n) \\
f = o(g) &\approx f(n) < c \cdot g(n) \\
f = \Omega(g) &\approx f(n) \geq c \cdot g(n) \\
f = \omega(g) &\approx f(n) > c \cdot g(n) \\
f = \Theta(g) &\approx f(n) = c \cdot g(n)
\end{aligned}$$

Theorem 1.1 (Master Theorem)**.** *If $T(n) = aT([n/b]) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then*

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

2 Polynomial Interpolation

Given a degree n polynomial $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$, the relationship between its values and coefficients can be represented by

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ & & \vdots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} \quad \text{(evaluation)}$$

where the matrix M is a *Vandermonde* matrix.

2.1 Fast Fourier Transform (FFT)

Definition 2.1 (Discrete Fourier Transform Matrix)**.** For polynomials of degree $< n$ (n is even; polynomials can be 0-padded), the Discrete Fourier Transform can be represented by the matrix

$$M_n(\omega) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ & & \vdots & & \\ 1 & \omega^j & \omega^{2j} & \cdots & \omega^{(n-1)j} \\ & & \vdots & & \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}$$

where $\omega = e^{2\pi i/n}$ is the nth root of unity.

Remark. $M_n(\omega)$ is an unitary matrix whose columns forms the *Fourier Basis*.

Lemma 2.1. $M_n^{-1}(\omega) = \frac{1}{n}\overline{M_n(\omega)} = \frac{1}{n}M_n(\omega^{-1})$

Lemma 2.2. $A(x) = A_{even}(x^2) + xA_{odd}(x^2)$

Remark. If A is evaluated at points $\pm\omega_0, \ldots, \pm\omega_{n/2-1}$, then $A_e(x^2)$ and $A_o(x^2)$ will only need to evaluate half the amount of points ($T(n) = 2T(n/2) + O(n) = O(n \log n)$).

Algorithm 1: Fast Fourier transform

Input: A coefficient vector, $\vec{a} = \langle a_0, \ldots, a_{n-1}\rangle$ and the nth root of unity, ω.
Output: $M_n(\omega)\vec{a}$

```
Function FFT(a⃗, ω):
    if ω = 1 then
        return a⃗
    else
        ⟨A_e(0), ..., A_e(n/2 − 1)⟩ ← FFT(⟨a_0, a_2, ..., a_{n−2}⟩, ω^2)
        ⟨A_o(0), ..., A_o(n/2 − 1)⟩ ← FFT(⟨a_1, a_3, ..., a_{n−1}⟩, ω^2)
        for j := 0 to n/2 − 1 do
            A(j) ← A_e(j) + ω^j A_o(j)
            A(j + n/2) ← A_e(j) − ω^j A_o(j)
    return ⟨A(0), ..., A(n − 1)⟩
```

2.2 Polynomial Multiplication

Algorithm 2: Fast Polynomial Multiplication

Input: Coefficient vectors, a and b, and the nth root of unity, ω.
Output: The coefficient vector of $A(x)B(x)$

$\hat{\vec{a}} \leftarrow M_n(\omega)\vec{a}$ (FFT)
$\hat{\vec{b}} \leftarrow M_n(\omega)\vec{b}$
for $i = 0$ *to* $n - 1$ **do**
  $\hat{c}_i \leftarrow \hat{a}_i \hat{b}_i$
return $\frac{1}{n} M_n(\omega^{-1})\hat{\vec{c}}$ (inverse matrix)

2.3 Cross-Correlation

Definition 2.2 (Cross-Correlation)**.** $corr(\vec{x}, \vec{y})[k] = \sum x_i y_{i-k}$, which measures similarity.

Algorithm 3: Cross-Correlation

Input: Two signal vectors, $\vec{x}$ and $\vec{y}$.
Output: $corr(\vec{x}, \vec{y})$

$X(t) \leftarrow x_{m-1} + x_{m-2}t + \cdots + x_0 t^{m-1}$
$Y(t) \leftarrow y_0 + y_1 t + \cdots + y_{n-1} t^{n-1}$
$Q(t) \leftarrow X(t)Y(t)$ (Fast Polynomial Multiplication)
return $\vec{q}$

3 Graphs

Definition 3.1 (Graph)**.** A graph is a pair $G = (V, E)$, typically represented by an adjacency matrix or an adjacency list.

Table 1: Graph representations.

	Space	Connectivity	getNeighbors(u)	DFS Runtime
Adjacency Matrix	$\Theta(\|V\|^2)$	$O(1)$	$\Theta(\|V\|)$	$\Theta(\|V\|^2)$
Adjacency List	$\Theta(\|V\| + \|E\|)$	$\Theta(degree(u))$	$\Theta(degree(u))$	$\Theta(\|V\| + \|E\|)$

3.1 Depth-First Search

Algorithm 4: Depth-first search

Input: V, E of directed graph G.

```
Function DFS(V, E):
    n ← |V|
    clk ← 1
    visited ← boolean[n]
    preorder, postorder = int[n]
    for v ∈ V do
        if !visited[v] then
            EXPLORE(v)

Function EXPLORE(v):
    visited[v] ← True
    preorder[v] ← clk++
    for (v, w) ∈ E do
        if !visited[w] then
            EXPLORE(w)
    postorder[v] ← clk++
```

Remark. Preorder-postorder intervals are either nested or disjoint.

Definition 3.2 (Back Edge)**.** A back edge is an non-tree edge that goes from a descendant vertex to an ancestor vertex.

Remark. postorder$[u] \leq$ postorder$[v]$ *iff* (u, v) is a back edge.

Remark. G is a cycle *iff* it contains a back edge.

3.1.1 Applications of DFS

Algorithm 5: Topological sort.

Input: A directed cyclic graph G.

Output: An ordered list of V such that u_i comes before v_i for all $(u_i, v_i) \in E$ (i.e., ordered by decreasing dependency).

```
post ← DFS-visited vertexes ordered by postorder visits
return reverse(post)
```

Definition 3.3 (Strongly Connected Component)**.** A SCC is a maximal partition of a directed graph in which every vertex is reachable from every other vertex.

$$u \text{ is in sink SCC of graph } G \Leftrightarrow u \text{ is in source SCC of reverse graph } G$$
$$\Leftrightarrow$$

u is in source SCC if highest postorder number.