

Efficient Algorithms and Intractable Problems

Daniel Deng

1 Complexity Analysis

Definition 1.1 (Asymptotic Notations).

$$f = O(g) \approx f(n) \leq c \cdot g(n)$$

$$f = o(g) \approx f(n) < c \cdot g(n)$$

$$f = \Omega(g) \approx f(n) \geq c \cdot g(n)$$

$$f = \omega(g) \approx f(n) > c \cdot g(n)$$

$$f = \Theta(g) \approx f(n) = c \cdot g(n)$$

Theorem 1.1 (Master Theorem). *If $T(n) = aT(\lfloor n/b \rfloor) + O(n^d)$ 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 $A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$, the relationship between its values and coefficients can be represented by

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

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 n th root of unity.

Remark. $M_n(\omega)$ is a 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. $A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$

Remark. If A is evaluated at points $\pm\omega_0, \dots, \pm\omega_{n/2-1}$, 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, $\vec{a} = \langle a_0, \dots, a_{n-1} \rangle$ and the n th root of unity, ω .

Output: $M_n(\omega)\vec{a}$

```

1 Function FFT( $\vec{a}, \omega$ ):
2   if  $\omega = 1$  then
3     return  $\vec{a}$ 
4   else
5      $\langle A_e(0), \dots, A_e(n/2 - 1) \rangle \leftarrow \text{FFT}(\langle a_0, a_2, \dots, a_{n-2} \rangle, \omega^2)$ 
6      $\langle A_o(0), \dots, A_o(n/2 - 1) \rangle \leftarrow \text{FFT}(\langle a_1, a_3, \dots, a_{n-1} \rangle, \omega^2)$ 
7     for  $j := 0$  to  $n/2 - 1$  do
8        $A(j) \leftarrow A_e(j) + \omega^j A_o(j)$ 
9        $A(j + n/2) \leftarrow A_e(j) - \omega^j A_o(j)$ 
10  return  $\langle A(0), \dots, A(n-1) \rangle$ 

```

2.2 Polynomial Multiplication

Algorithm 2: Fast Polynomial Multiplication

Input: Coefficient vectors, a and b , and the n th root of unity, ω .

Output: The coefficient vector of $A(x)B(x)$

```

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

2.3 Cross-Correlation

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

Algorithm 3: Cross-Correlation

Input: Two signal vectors, \vec{x} and \vec{y} .

Output: $\text{corr}(\vec{x}, \vec{y})$

```

1  $X(t) \leftarrow x_{m-1} + x_{m-2}t + \dots + x_0t^{m-1}$ 
2  $Y(t) \leftarrow y_0 + y_1t + \dots + y_{n-1}t^{n-1}$ 
3  $Q(t) \leftarrow X(t)Y(t)$  (Fast Polynomial Multiplication)
4 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	$\Theta(V + E)$	$\Theta(\text{degree}(u))$	$\Theta(\text{degree}(u))$	$\Theta(V + E)$

3.1 Depth-First Search

Algorithm 4: Depth-first search

Input: V, E of directed graph G .

```

1 Function DFS( $V, E$ ):
2    $n \leftarrow |V|$ 
3    $clk \leftarrow 1$ 
4    $visited \leftarrow \text{boolean}[n]$ 
5    $preorder, postorder = \text{int}[n]$ 
6   for  $v \in V$  do
7     if  $\neg visited[v]$  then
8        $\text{EXPLORE}(v)$ 
9 Function EXPLORE( $v$ ):
10   $visited[v] \leftarrow \text{True}$ 
11   $preorder[v] \leftarrow clk++$ 
12  for  $(v, w) \in E$  do
13    if  $\neg visited[w]$  then
14       $\text{EXPLORE}(w)$ 
15   $postorder[v] \leftarrow clk++$ 

```

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 contains 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).

```

1  $post \leftarrow$  DFS-visited vertexes ordered by postorder visits
2 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.

3.2 Single-Source Shortest Paths (SSSPs)

Algorithm 6: Single-Source Shortest Path

Input: A directed graph G and a start vertex S .

Output: Two arrays $prev[|V|]$ (shortest-path predecessor) and $dist[|V|]$ (shortest-path distance).

```

1 Function BFS( $G, S$ ):
  | /* Must have uniform edge weights.   $O(|V| + |E|)$  runtime.          */
2 Function Dijkstra ( $G, S$ ):
  | /* Must have positive edge weights.   $O(|V| \log |V| + |E|)$  runtime if
  |   implemented using fibonacci heap.                                */
3 Function Bellman-Ford ( $G, S$ ):
  | /* Can have arbitrary edge weights.                                */

```

3.3 Minimum Spanning Tree

Algorithm 7: MST algorithms.

Input: A graph G and a starting vertex v .

Output: The minimum spanning tree T of G .

/* Use the cut property. */

```

1 Function Prim( $G, v$ ):
2   |  $T \leftarrow \emptyset$ 
3   |  $H \leftarrow$  new Fibonacci heap

```
