Efficient Algorithms and Intractable Problems

Daniel Deng

1 Complexity Analysis

Definition 1.1 (Partial Sums).

$$S_k = \sum_{n=1}^{k} a_n = \frac{k}{2}(a_1 + a_k) \qquad \text{(Arithmetic Series)}$$

$$S_k = \sum_{n=1}^{k} a_1(r)^n = a_1\left(\frac{1-r^k}{1-r}\right) \qquad \text{(Geometric Series)}$$

Definition 1.2 (Asymptotic Notations).

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

Remark. $O(i^n \mid i > 1) > O(n^j) > O(log^k n)$

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

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

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(`$\vec{a}$, ω`)`**:**
- **if** $\omega = 1$ **then**
  - return $\vec{a}$
- **else**
  - $\langle A_e(0), \ldots, A_e(n/2-1) \rangle \leftarrow \text{FFT}(\langle a_0, a_2, \ldots, a_{n-2} \rangle, \omega^2)$
  - $\langle A_o(0), \ldots, A_o(n/2-1) \rangle \leftarrow \text{FFT}(\langle a_1, a_3, \ldots, a_{n-1} \rangle, \omega^2)$
  - **for** $j := 0$ *to* $n/2 - 1$ **do**
    - $A(j) \leftarrow A_e(j) + \omega^j A_o(j)$
    - $A(j + n/2) \leftarrow A_e(j) - \omega^j A_o(j)$
- **return** $\langle A(0), \ldots, A(n-1) \rangle$

2.2 Applications of FFT

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)

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++
/* Preorder-postorder intervals are either nested or disjoint.   */
/* postorder[u] ≤ postorder[v] iff (u, v) is a back edge.        */
/* G contains a cycle iff it contains a back edge.               */
```

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) ∈ E (i.e.,
        ordered by decreasing dependency).
post ← DFS-visited vertexes ordered by postorder visits
return reverse(post)
```

Definition 3.2 (Strongly Connected Component)**.** A SCC is a maximal partition of a directed graph in which every vertex is reachable from every other vertex.

$$\begin{aligned} u \text{ is in sink SCC of graph } G &\Leftrightarrow u \text{ is in source SCC of reverse graph } G \\ &\Leftrightarrow \text{u is in source SCC if highest postorder number.} \end{aligned}$$

3.2 Single-Source Shortest Path

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 aribitrary edge weights.                               */
```

3.3 Minimum Spanning Tree

Algorithm 7: Minimum spanning tree.

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):
      /* Sequentially adds the closest neightbor of the running set.
         O(|E| + log |V|) runtime if implemented using Fibonacci heap.   */
2 Function Kruskal(G, v):
      /* Sequentially adds the shortest edge that does not create a
         cycle.  O(|E| log |V|) runtime if implemented using Union-Find
         data structure.                                                 */
```

4 Greedy Algorithm

Definition 4.1 (Greedy Algorithm)**.** A greedy algorithm is one that builds the solution iteratively using a sequence of local choices.

Algorithm 8: Example greedy algorithms.

Function `SCHEDULING`**:**

`/* Find the maximum set of jobs that can be completed within time by iteratively select the next job to have the smallest end time without conflicting existing schedule.` $O(n \log n)$ `runtime if sorting the collection of jobs first. */`

Function `HUFFMAN`**:**

`/* Find a prefix tree for prefix-free Huffman coding by iteratively combine the two least frequent elements of the alphabet and retrieve the order of the prefix tree accordingly.` $O(n \log n)$ `runtime if implemented with min-heap. */`

Input: A set of partitions $S = \{S_1, \dots, S_m\}$ that covers the universe $\{1, \dots, n\}$.
Output: The indices of the smallest sub-collection of S that covers the universe.

Function `SET-COVER`**:**

`/* Greedy search yields sub-optimal but competitive solution to the set-cover problem. If the optimal solution uses` k `sets, then the greedy solution uses at most` $k \ln n$ `sets. */`

$A \leftarrow \{1, \dots, n\}$
$B \leftarrow \emptyset$
while $|A| > 0$ **do**
  let $i \in [m] \setminus B$ be s.t. $|A \cap S_i|$ is maximum
  $A \leftarrow A \setminus S_i$
  $B \leftarrow B \cup i$
return B

5 Union-Find

Definition 5.1 (Amortized Analysis)**.** Suppose a data structure supports k operations. Then the amortized cost of each operation is t_i if for any sequence of operations with N_i of O_i operations, the total time is at most $\sum_{i=1}^{k} t_i N_i$.

Algorithm 9: Union-find (disjoint forest implementation).

```
/* O((m+n) log* n) runtime.                                    */
parent[1,...,n]
rank[1,...,n] // rank is defined as the height if no path compression
    occur.
Function MAKE-SET(x):
    parent[x] ← x
    rank[x] ← 0
Function FIND(x):
    if x = parent[x] then
        return x
    parent[x] ← FIND(parent[x]) // path compression
    return parent[x]
Function UNION(x, y):
    x ← FIND(x)
    y ← FIND(y)
    if x = y then
        return // no work needed
    if rank[x] > rank[y] then
        swap x and y
    parent[x] ← y if rank[x] = rank[y] then
        rank[y] ← rank[y] + 1
```

Remark. Union-Find Invariants:

- a tree rooted at x has $\geq 2^{r[x]}$ items;
- $(\forall x)$, if x is not a root, $r[p[x]] > r[x]$;
- the number of items of exactly rank k is $\leq \frac{n}{2^k}$.

6 Dynamic Programming

Definition 6.1 (Top-Down DP/Memoization)**.** Recursion + look-up table

Definition 6.2 (Bottom-Up DP)**.** Fill up the look-up table iteratively instead of recursively

Remark. Bottom-up DP sometimes have better memory

7 Linear Programming

Definition 7.1 (Linear Programming)**.** Linear programming (LP) describes a broad class of optimization tasks in which both the constraints and the optimization criterion are linear functions. The optimum of a linear program is achieved at a vertex of the convex feasible region.

Remark. A linear program does not have an optimum *iff* its feasible region is infeasible and/or unbounded.

Definition 7.2 (Simplex Method)**.** A standard greedy algorithm for solving LP by hill-climbing on vertices of the feasible region.

Remark. Solves real-life LP in polynomial time.

7.1 LP Conversion

1. (Maximization $\leftrightarrow$ minimization) multiply the coefficients of the objective function by -1
2. (Inequality $\to$ equality) $ax \leq b \to ax + s = b \mid s \geq 0$
3. (Equality $\to$ Inequality) $ax = b \to ax \leq b \wedge ax \geq b$
4. (Signed $\leftrightarrow$ unsigned) $x \leftrightarrow x^+ - x^- \mid x^+, x^- \geq 0$

7.2 Network Flow

Definition 7.3 (Flow)**.** Given a directed graph $G = (V, E)$ with capacities $c_e > 0$ on all edges. The flow f from source s to sink t satisfies the constraints:

1. For all $e \in E$,
$$0 \leq f_e \leq c_e$$
2. For all $e \in E \setminus \{s, t\}$,
$$\sum_{(w,u)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz} \quad \text{(conservation of flow)}$$

Remark. By the conservation principle,

$$\text{size}(f) = \sum_{(s,u)\in E} f_{su}$$

Algorithm 10: Ford-Fulkerson

```
// Simplex algorithm for solving max-flow problem.  Pseudopolynomial
   time complexity; O(|E|F)
```

Input: $G = (V, E), s, t$

Output: f

$f \leftarrow 0$

$E^f \leftarrow E \cup \{rev(e) \mid e \in E\}$

$G^f \leftarrow (V, E^f)$ `// residual graph`

while *t reachable from s in G^f* **do**

  Choose an (s, t) path in G^f and maximize flow along the edges of the path `// reverse flow cancels existing flow`

  $$c^f \leftarrow \begin{cases} c_{uv} - f_{uv} & [(u, v) \in E \wedge f_{uv} < c_{uv}] \\ f_{vu} & [(v, u) \in E \wedge f_{vu} > 0] \end{cases}$$

return f

Theorem 7.1 (Max-flow Min-cut Theorem)**.** *The size of the maximum flow in a network equals the capacity of the smallest (s, t)-cut (L, R) (total capacity of the edges crossing the cut)*

Remark. L contains all reachable vertices from s in the final residual G^f and R contains all remaining vertices.

7.3 Duality

Theorem 7.2 (Duality theorem)**.** *If a linear program has a bounded optimum, then so does its dual, and the two optimum values coincide.*

Definition 7.4 (Duality)**.** Given two forms of the same LP:

- Primal LP — $\underset{\vec{x}}{\operatorname{argmax}}\,\{\vec{c}^{\mathsf{T}}\vec{x} \mid A\vec{x} \leq \vec{b}\}$
- Dual LP — $\underset{\vec{y}}{\operatorname{argmin}}\,\{\vec{b}^{\mathsf{T}}\vec{y} \mid A^{\mathsf{T}}\vec{y} = \vec{c},\, \vec{y} \geq 0\}$

Then,

- Weak duality — $\text{OPT}(P) \leq \text{OPT}(D)$
- Strong duality — Primal LP is bounded and feasible $\implies$ Dual LP is bounded and feasible $\wedge$ $\text{OPT}(P) = \text{OPT}(D)$

Remark. Dual/Primal unbounded $\implies$ Primal/Dual infeasible.