Algorithm Design and Analysis (Fall 2022) Midterm Exam Cheat Paper

Xiangyuan Xue (521030910387)

Time Complexity

Definition of big O notation:

- $T(n) = O(g(n))$:
  $\exists C, n_0, \forall n > n_0 : T(n) \leq C \cdot g(n)$
- $T(n) = \Omega(g(n))$:
  $\exists C, n_0, \forall n > n_0 : T(n) \geq C \cdot g(n)$
- $T(n) = \Theta(g(n))$:
  $T(n) = O(g(n)) = \Omega(g(n))$

Master Theorem (Generalized)

Given $a \geq 1, b > 1, d \geq 0, w \geq 0$, suppose $T(n) = 1$ for $n < b$ and $T(n) = aT(\frac{n}{b}) + n^d \log^w n$, then

$$T(n) = \begin{cases} O(n^d \log^w n) & \text{if } a < b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \\ O(n^d \log^{w+1} n) & \text{if } a = b^d \end{cases}$$

Just let $w = 0$, it falls back to original version.

Karatsuba

Let $z = (a+b)(c+d)$, $ad + bc = z - ac - bd$, thus

$$\begin{aligned} x \cdot y &= \left(a \cdot 10^{\frac{n}{2}} + b\right) \cdot \left(c \cdot 10^{\frac{n}{2}} + d\right) \\ &= ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd \\ &= ac \cdot 10^n + (z - ac - bd) \cdot 10^{\frac{n}{2}} + bd \end{aligned}$$

where time complexity is $O(3^{\log_2 n}) \approx O(n^{1.6})$.

Median of medians

An algorithm for selecting k-th element in S:

1. Partition S evenly into $\frac{n}{5}$ groups and directly find their $\frac{n}{5}$ medians.
2. Pick median of these $\frac{n}{5}$ medians as pivot and divide S into L, M, R by comparing to pivot.
3. If $k \leq |L|$, select k-th element in L.
   If $|L| < k \leq |L| + |M|$, pivot is the answer.
   If $|L|+|M| < k$, select $k-|L|-|M|$-th element in R.

It holds that $T(n) = T(0.2n) + T(0.7n) + O(n)$, by induction we can prove its complexity is $O(n)$.

Closest Pair

An algorithm to find the closest pair in 2-D plane:

1. Sort the points S by x and draw a vertical line l with $\frac{n}{2}$ points falling on each side.
2. Recursively find the closest pair on each side, where the distance is δ_L and δ_R.
3. Let $\delta = \min\{\delta_L, \delta_R\}$, drop points further than δ from l and sort remaining points S' by y.
4. For each $a \in S'$, check 7 points above in S' to find the closest pair, which is the answer.

Correctness relies on the limit of δ. Since $T(n) = 2T(\frac{n}{2}) + O(n \log n)$, its complexity is $O(n \log^2 n)$.

Fast Fourier Transform

Suppose p's degree is $D-1$ and $\omega = e^{\frac{2\pi}{D} i}$, function $\text{FFT}(p, \omega)$ is described as follow:

1. If $\omega = 1$, return $\{p(1)\}$.
2. $p_e(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{D-2} x^{\frac{D-2}{2}}$.
   $p_o(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{D-1} x^{\frac{D-2}{2}}$.
3. $\{p_e(\omega^0), \ldots, p_e(\omega^{D-2})\} = \text{FFT}(p_e, \omega^2)$.
   $\{p_o(\omega^0), \ldots, p_o(\omega^{D-2})\} = \text{FFT}(p_o, \omega^2)$.
4. For $t = 0, 1 \ldots, D-1$, we have
   $p(\omega^t) = p_e(\omega^{2t}) + \omega^t \cdot p_o(\omega^{2t})$.
5. Return $\{p(\omega^0), p(\omega^1), \ldots, p(\omega^{D-1})\}$

Complexity $T(n) = 2T(\frac{D}{2}) + O(D) = O(D \log D)$, which makes polynomial multiplication $O(n \log n)$. Inverse transformation is just $\text{FFT}(r, \omega^{-1})$.

Topological Sort

Run DFS on DAG and record finishing time. Sort vertices by descending order of finishing time, which is also the topological order. Just append the vertex as soon as it is finished, which is $O(|V| + |E|)$.
To prove correctness, we claim that if $finish[v] > finish[u]$, there will be no edge (u, v). By DFS properties, (u, v) won't be tree edge, forward edge or cross edge, and it can't be back edge on DAG, which proves the claim and promises correctness.

Algorithm Design and Analysis (Fall 2022) Midterm Exam Cheat Paper

Xiangyuan Xue (521030910387)

Strongly Connected Component

We can find SCCs by DFS as follow:

1. Construct reversed graph G^R.

   DFS G^R and record finishing time.

2. Choose v with the largest finishing time.

   DFS from v in G and form new SCC.

3. Remove new SCC from both G and G^R.

   Repeat until G and G^R are empty.

Note that the vertex with largest finishing time in G^R is in tail SCC in G, and successive vertices in G won't be visited again because they are removed. This algorithm is proved to be $O(|V| + |E|)$.

Single Source Weighted Shortest Path

Dijkstra works without negative weights:

1. $T = \{s\}$, $dist[s] = 0$, $dist[v] = \infty$ $(\forall v \neq s)$.
   $dist[v] = w(s, v)$ $(\forall (s, v) \in E)$.

2. Find $v \notin T$ with smallest $dist[v]$.

   $T = T + \{v\}$.

3. $dist[u] = \min\{dist[u], dist[v] + w(v, u)\}$
   $(\forall (v, u) \in E)$.

   Repeat until $T = V$.

Correctness relies on step 2. Just prove such v is optimal. Simple realization takes $O(|V|^2+|E|)$. By binary heap it can be $O((|V|+|E|)\log|V|)$, and by Fibonacci heap it can be $O(|E| + |V|\log|V|)$.

Negative Weighted Shortest Path

Bellman-Ford is slower but works:

1. $dist[s] = 0$, $dist[v] = \infty$ $(\forall v \neq s)$.

2. For each $(u, v) \in E$, update $dist[v] = \min$ $\{dist[v], dist[u] + w(u, v)\}$

3. Repeat until none of $dist[v]$ is updated.

After k rounds, $dist[v]$ is the shortest distance of k-edge paths. It terminates as long as the shortest path includes no longer than $|V| - 1$ edges. Otherwise, a negative cycle exists and there is no shortest path, which can be figured out by Bellman-Ford. Time complexity is obviously $O(|V| \cdot |E|)$.

Minimum Spanning Tree

Kruskal and Prim are two typical greedy algorithms to solve MST. They are simple, so description is omitted. Both of them works with negative weights and can be proved by induction, replacement and contradiction.
Time complexity of Kruskal is $O(|E|\log|E|) = O(|E|\log|V|)$, with find-union set optimization.
Time complexity of Prim is $O(|E| + |V|\log|V|)$, with Fibonacci heap optimization.

Huffman Encoding

This greedy algorithm repeats $n-1$ rounds. In each round, it finds two minimized elements and merge them into a new element, which forms a hyper node on Huffman tree. It is $O(n \log n)$ with heap optimization. If elements are initially sorted, it can be $O(n)$ by maintaining two increasing lists.

To prove its correctness, reformulate the cost as

$$S = \sum_{k=1}^{n-1}(w_{k,x} + w_{k,y}) = \sum_{i=1}^{n} x_i d_i = \sum_{i \in C} w_i d_i$$

For any nodes w_i and w_j such that $w_i < w_j$ and $d_i < d_j$, swapping them will reduce S (Rearrangement Inequality). Therefore, S is minimum when nodes with smaller weight have larger depth.

Makespan Minimization

This problem is NP-hard, so consider an approximation algorithm LPT, where we:

- Choose the job with longest processing time.
- Insert the job into earliest finished machine.

It does not promise optimal solution S^*, but its solution S is no larger than $\frac{4}{3}$ of S^*.
To prove that, consider a stronger proposition. Suppose the shortest job is p_n, we have $3p_n \leq S^*$.
Assume $S^* < 3p_n$, we know every machine in optimal solution has no more than 2 jobs. However, LPT will find this optimal solution in this case, namely $S = S^* < \frac{4}{3}S^*$. Otherwise, we have $S \leq S^* + p_n \leq \frac{4}{3}S^*$. Now the ratio has been proved.