Analysis of Algorithms - Assignment 2

Zhenrui Zheng

Chinese University of Hong Kong, Shenzhen
225040512@link.cuhk.edu.cn

Contents

Problem 1: Optimal Information Transfer

Let's first elaborate on the form of the optimal solution, and then prove it. The optimal solution will have the following form:

1. For any server, all rows of info on it will have the same copying strategy (that is, two rows of info will not be copied to different servers);
2. Rows of info will either only be copied from A to B, or only from B to A;
3. For the cluster that receives info in 1, there will be at most one "hub", that is, it receives info from other servers in the same cluster. The sender cluster will not have such a hub;
4. For each server in the receiving cluster that does not send its own info to the hub (including the hub itself), it will receive info from every server in the sender (the other cluster).

1.1 Proof of Optimality

Hereafter, we refer to the i-th server in cluster A as A_i, and similarly, B_i; when not distinguishing the cluster a server belongs to, we refer to it as S; the j-th row of server S is referred to as r_j^S. In a certain copying strategy, let the set of all rows on a server S_i be denoted as $R^{S_i} = \{r_y^{A_x}, ..., r_y^{B_x}, ...\}$, and $R_A^{S_i} = \{r_y^{A_x}, ...\}$ be the set of all rows on S_i that are copied from A cluster.

Lemma 1. *For any given row r_j^S, it will not be copied more than twice (i.e., from S_0 to S_1, and then from S_1 to S_2).*

Proof. Obviously, r_j^S can be copied directly from S_0 to S_2 without affecting the result. □

Lemma 2. *For any server, all rows on it will have the same copying strategy.*

Proof. Suppose there are two rows of info, r_i and r_j, on the same server, which adopting different copy strategies to "meet" all rows from another cluster, with possibly different costs; obviously, we can always change the row with the higher cost to adopt the same strategy as the row with the lower cost, and still "meet" all rows from the other cluster, without making the answer worse. If the cost is the same, we can simply choose one of the two strategies. Therefore, we can always guarantee that all rows in the same server adopt the same copy strategy and still get some kind of optimal solution. □

Since we have already shown that all rows on the same server can adopt the same copy strategy, we will directly refer to "copy A_i to S" below.

Lemma 3. *If a server's rows are copied to another, we call it a "sender", and vice versa, it's a "receiver". We can always construct a scenario such that a server is not both a sender and a receiver.*

Proof. Assume that a server S_0 is copied to at least one other server $\{S_1, ..., S_l\}$ (being a "sender") and has information from some other servers $\{S'_1, ..., S'_l\}$ on itself (being a "receiver"), then we can always switch to copy $\{S'_1, ..., S'_l\}$ to any S_i, without incurring additional costs or affecting the solution (without losing any valid pair of rows). □

Corollary 1. *Without loss of generality, assume A_i is copied to B_i. By definition, A_i is a sender and B_i is a receiver. To pair with all A's rows, every server in A must be copied to B_i. Thus, all A_i become senders and cannot be receivers. Thus, rows can only be copied from the A cluster to the B cluster. Similarly, if any B_i is copied to A_i, then only B is copied to A.*

Lemma 4. *By definition, if a server S has copies from other servers in the same cluster, we call it a "hub". We can always construct a cluster such that there is at most one "hub" in the same cluster.*

Proof. Assume S_i and S_j are from the same cluster and are both "hub", then they are both receivers by definition, and thus have rows from all other cluster servers. Then, we can always transfer rows from the same cluster server on S_i (except S_i itself) to S_j, without additional cost or affecting the solution, and thereby exempt S_i from its "hub" identity. □

Corollary 2. *By definition, a "hub" is a receiver, so sender cluster will have no "hub".*

Corollary 3. *Examine each server in the receiver cluster, which will be one of three types: 1. a "hub", 2. a server that copied itself to the "hub", 3. an individual server that doesn't copy itself to the "hub". Obviously, all servers in the sender cluster will copy themselves to each individual server and hub.*

Lemma 5. *The "hub" will always be the server with the most rows in the receiver cluster.*

Proof. Obviously, if the "hub" is not the server with the most rows, we can transfer the "hub" identity (and all rows on it) to the server with the most rows, which does not affect the properties of the solution and does not increase the cost. □

Therefore, we can select the server with the most rows in clusters A and B as the "hub", respectively, and enumerate the servers in the receiver cluster (i.e., the cluster where the "hub" is located) to check whether they should be copied to the hub (i.e., which is higher: the cost of copying itself to the hub or the cost of copying all sender clusters to it). A special case is when every row of the sender cluster is copied to every server in the receiver cluster, i.e., there is no "hub", which is already included in this solution. Obviously, the time complexity of this solution is $O(n + m)$.

Problem 2: Path Counting

This is a classic combinatorial mathematics problem that can be solved using the principle of inclusion-exclusion + combinations. We can transform the problem of finding the total number of paths from (1,1) to (n,n) that do not pass through any obstacles into finding the total number of paths from (1,1) to (n,n) and subtracting the number of paths that pass through any obstacles.

Let A_i denote the set of paths that pass through the i-th obstacle, then the latter is $\bigcup A_i$, which can be calculated; if we denote the set of paths that do not pass through any obstacles as A_0, then the answer is $|A_0| - |\bigcup_{i=1}^{k} A_i|$.

Using the principle of inclusion-exclusion, $\bigcup_{i=1}^{k} A_i$ can be expressed as:

$$\begin{aligned}\left|\bigcup_{i=1}^{k} A_i\right| &= \sum_{i=1}^{k} |A_i| - \sum_{1\le i<j\le k} |A_i \cap A_j| + \sum_{1\le i<j<l\le k} |A_i \cap A_j \cap A_l| - \cdots + (-1)^{k+1}|A_1 \cap A_2 \cap \cdots \cap A_k| \\ &= \sum_{m=1}^{k} (-1)^{m-1} \sum_{a_i<a_{i+1}} \left|\bigcap_{i=1}^{m} A_{a_i}\right|\end{aligned}$$

In which, $A_{a_1} \cap A_{a_2} \cap ... \cap A_{a_m}$ is the path that passes through obstacles a_1 a_m simultaneously. The count is the product of the path counts from each obstacle A_{a_i} to the next $A_{a_{i+1}}$ (including the start and end points).

And the number of paths from one point (x_1, y_1) to (x_2, y_2) is $\binom{x_2-x_1+y_2-y_1}{x_2-x_1}$. Here, if $x_2 < x_1$ or $y_2 < y_1$, the count is 0. This is because you can only move down or right, and you need to move a total of $x_2 - x_1 + y_2 - y_1$ steps, so you need to choose $x_2 - x_1$ steps to move down and the other $y_2 - y_1$ steps to move right. In fact, doing $\binom{x_2-x_1+y_2-y_1}{y_2-y_1}$ is the same.

So we can simply enumerate the cases (obstacles passed) and calculate their contribution to the answer, which is:

$$(-1)^{m-1} \sum_{a_i<a_{i+1}} \left|\bigcap_{i=1}^{m} A_{a_i}\right|$$

However, since n is large, $\binom{x_2-x_1+y_2-y_1}{x_2-x_1}$ will be calculated modulo p. We can pre-process the factorials $\text{fact}(i) \pmod p$ and their modular multiplicative inverses $\text{inv}(i) \pmod p$ for 1 to $2n$, and then express $\binom{x_2-x_1+y_2-y_1}{x_2-x_1}$ as $\text{fact}(x_2-x_1+y_2-y_1) * \text{inv}(x_2-x_1) * \text{inv}(y_2-y_1)$ $\pmod p$.

Here, the multiplicative inverse of x modulo p is the solution to $ax \equiv 1 \pmod p$. According to Fermat's Little Theorem, we have $a^{-1} \equiv a^{p-2} \pmod p$, where the latter can be computed by exponentiation by squaring in $O(\log p)$ time.

The total time complexity includes preprocessing factorials and multiplicative inverses from 1 to $2n$, which is $O(n)$, and counting for each path combination, i.e., the number of combinations for obstacles, which is $O(2^k)$, that sums to $O(n + 2^k)$.

Problem 3: Array Flipping

Obviously, if the length of the reversed contiguous subsegment is odd, the final sum will not change (this is because the parity of each number before and after the reversal remains the same). When the reversal length is even, i.e., $(r - l + 1) \bmod 2 = 0$, the increment to the answer will be $\delta = \sum_{i=l, i \bmod 2=1}^{r} a_i - \sum_{i=l, i \bmod 2=0}^{r} a_i$. Therefore, we want to find the largest increment $\max \delta$ over all possible l and r.

However, the complexity of naively iterating through l and r is $O(n^2)$. We can achieve a single traversal by splitting the increment δ into a form separated by l and r.

$$
\begin{aligned}
\max_{l,r} \delta &= \max_r \max_l \left(\sum_{i=l, i \bmod 2=1}^{r} a_i - \sum_{i=l, i \bmod 2=0}^{r} a_i \right) \\
&= \max_r \max_l \left[\left(\sum_{i=0, i \bmod 2=1}^{r} a_i - \sum_{i=0, i \bmod 2=1}^{l-1} a_i \right) - \left(\sum_{i=0, i \bmod 2=0}^{r} a_i - \sum_{i=0, i \bmod 2=0}^{l-1} a_i \right) \right] \\
&= \max_r \left[\left(\sum_{i=0, i \bmod 2=1}^{r} a_i - \sum_{i=0, i \bmod 2=0}^{r} a_i \right) \max_l \left(- \sum_{i=0, i \bmod 2=1}^{l-1} a_i + \sum_{i=0, i \bmod 2=0}^{l-1} a_i \right) \right] \\
&= \max_r \left[p_r \min_l p_{l-1} \right]
\end{aligned}
$$

Where $p_i = \sum_{j=0, j \bmod 2=1}^{i} a_j - \sum_{j=0, j \bmod 2=0}^{i} a_j$ is the difference of the prefix sum of the numbers with odd indices and even indices, which can be pre-processed in $O(n)$ time.

Then, we can iterate through r while recording the $\min p_i$ for even i and odd i separately, and updating the maximum δ accordingly. The time complexity is $O(n)$. Thus the total time complexity is $O(n)$.

Problem 4: Cargo Port Assignment

We can transform this problem into a classic dynamic programming problem by expanding the unloading schedule of the i-th ship into n sub-conditions. The scheme is as follows: let the unloading arrangement for the i-th ship be that at least n_i days are spent unloading within days x_i to y_i. We can expand this to: for $j \in [x_i, y_i]$, within days x_i to j, at least $\max(0, j - (y_i - n_i))$ days are spent unloading.

Therefore, we denote $dp(i)$ as the minimum number of unloading days required up to day i (inclusive). The state transition is:

$$dp(j) = \max_{i, j \in [x_i, y_i]} \{dp(x_i - 1) + \max(0, j - (y_i - n_i))\}$$

This requires that when $dp(j)$ is used, it has already been calculated (and will not be updated hereafter). We can guarantee this by sorting all ships by x_i and processing them in order.

It should be noted that whenever the case of a ship i is processed, we also need to update $dp(j)$ for $j \in [x_{i-1}, x_i - 1]$:

$$dp(j) = \max(dp(j-1), dp(j))$$

To ensure that the ans before x_i is correct.

Since we need to update all ships a total of $\sum(n_i) = O(L \max(y))$ times (Equation 1), and post-process the states before all y_i for $O(\max(y))$ (Equation 2), and also pre-sort all ships for $O(L \log L)$, the final time complexity will be $O(L \max(y) + L \log L)$.