A Simple Introduction to CDQ Divide and Conquer

Jan 31, 2020

tags: `icpc` `algorithm` `divide-and-conquer` `cdq` `offline-techniques`

Outline

What is CDQ D&C?

CDQ D&C is a offline technique invented by a China competitive programmer (called CDQ, obviously). CDQ D&C is almost same to normal D&C. The biggest difference between them is that suppose we cut a interval $[l, r)$ into $[l, m), [m, r)$. In normal D&C, the elements from $[l, m)$ won't influence $[m, r)$; In CDQ D&C, elemenst from $[l, m)$ **will** influence $[m, r)$. Therefore, the framework of CDQ D&C usually looks like this.

1. Divide $[l, r)$ into $[l, m), [m, r)$.
2. Solve two subproblems $[l, m), [m, r)$.
3. Calculate the influence of $[l, m)$ on $[m, r)$.
4. Merge $[l, m), [m, r)$.

As the introduction above might be to abstract, let's dive into problems below to get more "feeling" about it. We'll abbreviate CDQ D&C to CDQ.

Example problems

ZJc571 - Three-dimensional partial order problem (Chinese)

Problem description

You're given a set of unique three-dimension points S with size n. For each point $P_i = (x_i, y_i, z_i) \in S$, please calculate the number of points $P_j \in S$ such that $x_i < x_j, y_i < y_j, z_i < z_j$.

$1 \le n \le 10^5, 1 \le x_i, y_i, z_i \le 10^5$

Problem analysis

First, this problem is solvable with any comparator with that have transitivity ($<, >, \le, \ge \ldots$). Formally, they're called partial order. This math version of this problem is $\sum_{x_i<x_j, y_i<y_j, z_i<x_j} 1$.

This problem is a classic problem that CDQ can deal with. Let's look at an easier problem first: Given an array A, please calculate the number of inversion pairs. There're lots of ways to do it, one of it (which we will use) is given here.

What's the connection between counting inversion pairs with the original problem? Counting inversion pairs is a Two-dimensional partial order problem! For every element A_i at position i, if we view it as (i, A_i), then number of inversion paris that ends with i is exactly number of (j, A_j) such that $j < i, A_j > A_i$.

So now we know how to solve Two-dimensional partial order problem. Then, we can solve Three-dimensional partial order problem by reducing it to Two-dimension using CDQ.

Problem solution

We reduce it by the following steps:

1. Sort one of the axes. Let's sort x axis here.
2. Suppose that we'll dealing with $[l, r)$. Split it into two subproblems $[l, m), [m, r)$ and deal with it recursively.
3. Note that "calculate influence of $[l, m)$ to $[m, r)$" is a Two-dimensional partial order problem because:
    1. We only need to consider whether each element in $[l, m)$ is a part of answer of each element in $[m, r)$ or not. This implies that order of x_i in $[l, m)$ is not important.
    2. x axis in $[l, m)$ is greater $[m, r)$ i.e. $\min_{i \in [l,m)} x_i \ge \max_{i \in [m,r)} x_i$, we successfully reduced the original problem to Two-dimension version.
4. Solve the Two-dimensional partial order problem by sorting $[l, r)$ with y_i and BIT.

By appling Master Theorem to $T(N) = 2T(\frac{N}{2}) + O(N \log N)$ we can get the complexity $O(N \log^2 N)$. There're serveral important details given in the code.

▶ code

BZOJ1176 - Mokia (Chinese)

Problem description

You're given a square matrix A with size $W \times W$, all elements in A are initialized with S. You need to handle Q operations of two kinds:

- $1\ x\ y\ a$: Increase $A_{x,y}$ by a.
- $2\ x_1\ y_1\ x_2\ y_2$: Calculate the sum of submatrix A' with (x_1, y_1) being the upper-left corner and (x_2, y_2) being the bottom-right corner.

$1 \leq W \leq 2 \times 10^6, 1 \leq Q \leq 10^4$

Problem analysis

This problem doesn't seem to have any connection to CDQ. Let's try to transform it.

1. the second operation can be calculated using four prefix sum.
2. Prefix sum is similar to partial order problem: prefix sum at (x, y) is $\sum_{i \leq x, j \leq y} A_{i,j}$.

Using the observation above, we can view the two operations in this problem as elements (t, x, y) (t is the time of the operation), and solve it with the method silimar to the previous problem.

Problem solution

- For type 1 operations, create element (t, x, y) with value a.
- For type 2 operations, create element $(t, x_2, y_2), (t, x_1 - 1, y_2), (t, x_2, y_1 - 1), (t, x_1 - 1, y_1 - 1)$.

For (t, x, y), $\sum_{(k,i,j)|k<t, i \leq x, j \leq y} A_{i,j}$ is exactly the prefix sum at time t. The complexity is $O(Q \log Q \log W)$.

See code for more details.

▶ code

BZOJ3295 - Dynamic Inversion Pairs (Chinese)

Problem description

You're given N and A, permutation of $1, 2, \ldots, N$. You need to support M operations, where in every operation you need to calculate number of inversion pairs and delete an element from the A. A inversion pair $(i, j), 1 \leq i, j, \leq N$ is a pair that satisfies $i < j$ and $A_i > A_j$.

$1 \leq N \leq 10^5, 1 \leq M \leq 5 \times 10^4$

Problem analysis

We already know that counting inversion pairs is a Two-dimensional partial order problem. However, in this problem we need to support "delete" operations. This can be handled by adding a dimension: valid time of the element. After adding it, this problem is reduced into a Three-

dimensional partial order problem, and can be solved with similar ways from above.

Problem solution

For every delete operations, add element (t, val, pos). Then, for element (t, val, pos), $\sum_{(k,i,j)|k>t,i>val,j<pos} 1 + \sum_{(k,i,j)|k>t,i<val,j>pos} 1$ is exactly the number of inversion pairs related to the element at pos. The two terms can be calculated using two CDQs, and the time complexity will be $O(N \log^2 N)$. More details in code.

▶ code

CF1093E - Intersection of Permutations

Problem description

You are given two permutations a, b, both consisting of N elements. You need to support M operations of two kinds:

- $1\ l_a\ r_a\ l_b\ r_b$: calculate the number of values which appear in both segment $[l_a, r_a]$ of positions in permutation a and segment $[l_b, r_b]$ of positions in permutation b.
- $2\ x\ y$: swap values on position x and y in permutation b.

$2 \leq N \leq 2 \times 10^5, 1 \leq M \leq 2 \times 10^5$

Problem analysis

As both sequence are permutations from $[1, N]$, we can given each element $i \in [1, N]$ a "coordinate": (position of i in a, position of i in b). Then, type 1 operation can be viewed as "prefix sum" from (l_a, l_b) to (r_a, r_b), and type 2 operations can be viewed as modifying values at position (x, y). Then, this problem is similar to Mokia, which can be solved by CDQ.

Problem solution

The time complexity is $O(M \log M \log N)$. Details are given in the code.

▶ code

TIOJ1921 - ATM2 (Chinese)

Problem description

You want to make a fortune by purchasing, selling, and of course, using cash printing machine (CPM). Now, every day is divided to three parts: morning(9~12), afternoon(13~18), and night(19~22). In the morning, you can sell CPM that you've purchased before. In the afternoon, you can use your CPM to print cash. At night, you can purchase a new CPM. Note that you cannot have two CPM at the same time i.e. before purchasing a new CPM, you must sell your old CPM.

You are given information of N CPMs. For the i^{th} CPM, it can only be bought at day D_i with price P_i, and it can be sold with price R_i. It can print G_i cash every day. Now you're given C

dollars in the beginning, please maximize the cash you can earn in D days i.e. maximize the cash you can have at 12 o'clock of day $D+1$.

$N \leq 10^5, C, D, G_i \leq 10^9, D_i \leq D, R_i < P_i \leq 10^9$

Problem analysis

This problem is actually a DP problem, and can be solved in $O(N \log N)$ using convex-hull optimization. Let dp_i represent the maximum cash you can have after buying the i^{th} machine, then

$$dp_i = \max_{0 \leq j < i, dp_j \geq 0} \{dp_j + (D_i - D_j - 1) \times G_j + R_j - P_i\}$$

, which can be rewritten to:

$$dp_i = \max_{0 \leq j < i, dp_j \geq 0} \{G_j D_i + dp_j + R_j - (D_j + 1) \times G_j\} - P_i$$

If we view $a_j = G_j$, $b_j = dp_j + R_j - (D_j + 1) \times G_j$, then:

$$dp_i = \max_{0 \leq j < i, dp_j \geq 0} \{a_j D_i + b_j\} - P_i$$

which is exactly the form of convex-hull optimization. Here's the code using dynamic convex hull to solve it:

▶ code

However, we can solve this problem with CDQ, which is easier if you forgot how to write dynamic convex hull during contest :).

Problem solution

For dp values in $[l, r)$:

1. Calculate dp values recursively in $[l, m)$.
2. Use dp values in $[l, m)$ to update $[m, r)$.
3. Calculate dp values recursively in $[m, r)$.
4. Merge two convex hull: One from $[l, m)$, and another from $[m, r)$. This can be easily done with a `std::vector` as the slope is increasing.

The complexity is $O(N \log N)$ as the merging part only takes $O(N)$. See details in code.

▶ code

More problems

- CF293E - Close Vertices
- BZOJ4836 - Binary Operation

A Simple Blog

robert881003@gmail.com

A blog for new year resolution.