Review For Point Count Problem And A Devide And Conquer Trick

陈小羽

2019 年 4 月 2 日

目录

1 Preliminary

1.1 点/向量 (point/vector)

向量空间中的点和向量其实是相同的概念. n 维空间中的点 v 可以用 n 元组 $(v_1, v_2, v_3, \cdots, v_n)$ 表示. 特别的, 为了简化, 一维空间中的点 x 直接使用 x 表示而省略括号. 我们可以将高维度的点看成许多一维空间中的点的元组.

```
struct Point {int w[d], id; bool inS;}; // d 是维度
```

1.2 点集 (point set)

n 维向量空间是包含无限个点的集合, 任意多个 n 维向量组成的集合都是 n 维向量空间的子集. 为了方便描述 n 维向量空间的子集, 引入记号 $\{x|P(x)\}$ 来表示所有满足条件 P 的点组成的集合. $|S|$ 表示集合 S 中的元素个数. 本文中用到的集合为可重复集, 有特殊情况会专门说明.

```
Point S[n];// 集合大小为n
```

1.3 n 维数点问题

1.3.1 n 维数点问题的一般形式 (ND Point Count Problem ⇒ PC)

- input: n 维空间中的两个点集, S, Q.
- output: 对 $\forall a \in Q$, 计算出 $\mathrm{PC}(a) = |\{b \in S| \bigwedge_{i=1}^{n} a_i \le b_i\}|$.

1.4 n 维偏序问题

1.4.1 n 维偏序问题的一般形式 (ND Partial Order Count Problem ⇒ POC)

- input: n 维空间中的两个点集, S, Q.
- output: 计算出 $\mathrm{POC} = |\{(a, b)|a \in S \wedge b \in Q \wedge \bigwedge_{i=1}^{n} a_i \le b_i\}|$.

1.5 数点问题和偏序问题的关系

很容易发现, 数点问题和偏序问题存在如下关系:

$$\mathrm{POC} = \sum_{x \in Q} \mathrm{PC}(x)$$

略线性的部分, 这篇文章中给出的方法, 在这两个问题上均能取得相同的复杂度.

1.6 树状数组 (ArrayTree)

树状数组是处理上面两类问题时常用的数据结构. 这里只需要知道树状数组可以用来维护一个一维点的集合 S, 并且具有如下能力:

- $\mathrm{Insert}(x)$: 将一个一维点 x 加入其维护的点集中, $\mathcal{O}(\log \mathrm{ub})$.
- $\mathrm{Query}(x)$: 返回 $|\{y \in S | y \leq x\}|$, $\mathcal{O}(\log \mathrm{ub})$.
- 其中, ub 表示 S 中点的坐标的变化范围的上界.

```
// const int n = |S|;
int lowbit (int x) { return x & (-x); }
void Init(int *w, int ub) {memset(w, 0, sizeof(int) * ub);}
void Insert (int *w, int x, int ub) { for (; x <= ub; x += lowbit(x)) ++w[x]; }
int Query (int *w, int x) {
    int ret = 0;
    for (; x >= 1; x -= lowbit(x)) ret += w[x];
    return ret;
}
```

2 求解方法

P[i].inS 表示 P[i]$\in S$, 否则 P[i]$\in Q$.

2.1 1 维数点问题

```
int cmp1 (const Point &a, const Point &b) { return a.w[1] < b.w[1]; }
int cmpid (const Point &a, const Point &b) { return a.id < b.id; }
// P: point set, PC: answer
void PC_1D(Point *P, int *PC, int l, int r) {
    sort(P+l, P+r+1, cmp1); // 按第一维排序
    for (int i = l, cnt = 0; i <= r; i++)
        if (P[i].inS) ++cnt;
        else PC[P[i].id] = cnt;
}
```

complexity: $\mathcal{O}(n \log n)$.

2.2 2 维数点问题

```
int cmp2 (const Point &a, const Point &b) { return a.w[2] < b.w[2]; }
// P: point set, PC: answer, w: ArrayTree, ub: upperbound for x
void PC_2D(Point *P, int *PC, int *w, int ub, int l, int r) {
    sort(P+l, P+r+1, cmp2); // 按第二维排序
    for (int i = l; i <= r; i++) {
        if (P[i].inS) Insert(w, P[i].w[1], ub);
        else PC[P[i].id] = Query(w, P[i].w[1]); // 存在相同元素时需要修改
    }
}
```

complexity: $\mathcal{O}(n \log n)$.

2.3 3 维数点问题

- 按第三维排序
- 递归解决 $[l, m]$.

- 计算 $[l, m]$ 的 S 中的点对 $[m+1, r]$ 的 Q 中的点的贡献. 因为第三维的相对顺序固定了, 所以问题退化为了一个二维的数点问题 (S, Q 和原问题不一样).

- 递归解决 $[m+1, r]$.

```
int cmp3 (const Point &a, const Point &b) { return a.w[3] < b.w[3]; }
void
PC_3D(Point *P, Point *P_aux, int *PC, int *PC_aux, int *w, int ub, int l, int r) {
    if (l == r) return;
    Init(w, ub);
    sort(P+l, P+r+1, cmp3); // 按第三维排序
    int mid = (l + r) >> 1;
    PC_3D(P, P_aux, PC, PC_aux, w, ub, l, mid);
    for (int i = l; i <= r; i++) {
        P_aux[i] = P[i];
        if (i >= mid+1) P_aux[i].inS = true;
    }
    PC_2D(P_aux, PC_aux, w, ub, l, r);
    for (int i = mid+1; i <= r; i++) if (!P[i].inS) PC[P[i].id] += PC_aux[P[i].id];
    PC_3D(P, P_aux, PC, PC_aux, w, ub, mid+1, r);
}
```

complexity: $T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + \mathcal{O}(n \log n) = \mathcal{O}(n \log^2 n)$.

2.4 n 维数点问题

按 3 维数点问题的思路, 可以不停的利用分治策略, 来将 d 维的问题转化为 $d-1$ 维上的问题 (对 2 维一样成立, 这样可以不用树状数组). 根据主定理, $T(n, d) = T(n, d-1) \log n$. complexity: $T(n, d) = \mathcal{O}(n \log^{d-1} n)$.