分治

杜鼎勋

天津大学

2023 年 8 月 12 日

目录

目录

分治简介

Description

分治，字面上的解释是“分而治之”，就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。

大概的流程可以分为三步：分解 -> 解决 -> 合并。

1. 分解原问题为结构相同的子问题。
2. 分解到某个容易求解的边界之后，进行求解。
3. 将子问题的解合并成原问题的解。

以归并排序为例

Description

归并排序

```
void mergesort(数组a){
    if (a大小为1){
        return;
    }
    mergesort(a的左半部分);
    mergesort(a的右半部分);
    合并两个有序数组成一个有序数组
}
```

以快速排序为例

Description

快速排序

```
void sort(数组a){
    if (a大小为1){
        return;
    }
    选择一个中间值mid
    把a分成两部分，把不大于mid的放在左部分，大于mid的放在
        右部分
    sort(a的左部分);
    sort(a的右部分);
}
```

以二叉树的遍历为例

Description

1. 先序遍历
先序遍历首先访问根结点然后遍历左子树，最后遍历右子树。
2. 中序遍历
中序遍历首先遍历左子树，然后访问根结点，最后遍历右子树。
3. 后序遍历
后序遍历首先遍历左子树，然后遍历右子树，最后访问根结点。

分治 FFT

Description

给出 n 个形如 $a \cdot x + b$ 的多项式，求他们的乘积

分析

FFT 的时间复杂度与多项式长度正相关，如果依次计算乘法复杂度是 $O(n^2 logn)$
采用分治思想，分别计算左半边和右半边，再将左半部分与右半部分乘起来，一共有 $logn$ 层，每层复杂度是 $O(n^2 logn)$，总复杂度为 $O(nlog^2 n)$

目录

线段树分治

Description

对于一类有插入、删除（撤销插入）和整体查询操作的题目，可以考虑用线段树分治解决，也叫时间分治。
线段树分治是一种离线做法，处理的是某种修改对询问的影响，这种影响可以独立也可以不独立。关键是把一个修改看成一个区间，每个询问是一个叶子。修改是在线段树上打标记。这里把询问看做一个时间点，而修改对应一个时间区间的修改。
做法：在线段树上记录要操作的时间区间，dfs 依次执行这一操作，一直到根节点来回答询问，离开时将其撤销。

例题

Description

有一个 n 个节点的图
有 m 条边会出现后消失，第 i 条边的存在时间是 $[l_i, r_i]$
有 k 次询问，每次询问在时刻 t 点 i 所在连通块的大小。

例题

Description

有一个 n 个节点的图
有 m 条边会出现后消失，第 i 条边的存在时间是 $[l_i, r_i]$
有 k 次询问，每次询问在时刻 t 点 i 所在连通块的大小。

分析

如果没有有效时间的话，就是并查集的裸题。
那么一次加边操作本质上影响的是 $[l_i, r_i]$ 这一时间段的答案。所以我们采用线段树分治，将每个操作都放到线段树的节点上，那么这个操作最多放在 $logn$ 个节点上。最后我们对这棵线段树进行 dfs，每次进入这个点就执行对应节点上的合并操作，一直到叶子节点回答询问，离开时撤销（使用可撤销并查集）。

FJOI2015 火星商店问题

Description

火星上的一条商业街里按照商店的编号 $1 \sim n$，依次排列着 n 个商店。每种商品都用一个非负整数 val 来标价。
对于给定的按时间顺序排列的事件，计算每个购物的火星人的在本次购物活动中最喜欢的商品，即输出 $val \bigoplus x$ 的最大值。这里所说的按时间顺序排列的事件是指以下两种事件：
0 s v，表示编号为 s 的商店在当日新进一种标价为 v 的商品。
1 l r x d，表示一位火星人当日在编号在 $[l, r]$ 的商店购买 d 天内的商品，该火星人的喜好密码为 x。

FJOI2015 火星商店问题

Description

火星上的一条商业街里按照商店的编号 $1 \sim n$，依次排列着 n 个商店。每种商品都用一个非负整数 val 来标价。
对于给定的按时间顺序排列的事件，计算每个购物的火星人的在本次购物活动中最喜欢的商品，即输出 $val \oplus x$ 的最大值。这里所说的按时间顺序排列的事件是指以下两种事件：
0 s v，表示编号为 s 的商店在当日新进一种标价为 v 的商品。
1 l r x d，表示一位火星人当日在编号在 $[l, r]$ 的商店购买 d 天内的商品，该火星人的喜好密码为 x。

分析

本题询问区间有两个，一个是商店的区间，一个是时间的区间。考虑用线段树分治，同样把询问插入到对应线段树时间区间上，同样的递归处理与回溯撤销，这样能处理掉时间的区间。对于商店的区间，可以用可持久化 trie 树来做异或最大值。

目录

例题

Description

给出一个长度为 n 的数组 a，让你从中选出若干个数，选择的数不能相邻，问选的数的和最大是多少

例题

Description

给出一个长度为 n 的数组 a，让你从中选出若干个数，选择的数不能相邻，问选的数的和最大是多少

考虑 dp

$f_{i,0}$ 表示考虑了前 i 个数，不选第 i 个数的和最大是多少
$f_{i,1}$ 表示考虑了前 i 个数，选第 i 个数的和最大是多少

```
for (int i=1;i<=n;i++){
    f[i][0]=max(f[i-1][1],f[i-1][0]);
    f[i][1]=f[i-1][0]+a[i];
}
```

例题

Description

给出一个长度为 n 的数组 a，多次询问，每次让你从 $a_l \sim a_r$ 中选出若干个数，选择的数不能相邻，问选的数的和最大是多少。带有修改操作。

考虑动态 dp

$f_{i,0}$ 表示考虑了前 i 个数，不选第 i 个数的和最大是多少
$f_{i,1}$ 表示考虑了前 i 个数，选第 i 个数的和最大是多少
每次转移只转移了一位
如果我们能知道左半部分、右半部分的一些 dp 信息，能否将其合并呢

```
f[left][0]表示左半边右端点不选的和最大是多少
f[left][1]表示左半边右端点选的和最大是多少
f[right][0]表示右半边左端点不选的和最大是多少
f[right][1]表示右半边左端点选的和最大是多少
ans=max(f[left][0]+f[right][1],f[left][0]+f[right][0],f[
    left][1]+f[right][0])
```

例题

Description

给出一个长度为 n 的数组 a，多次询问，每次让你从 $a_l \sim a_r$ 中选出若干个数，选择的数不能相邻，问选的数的和最大是多少。带有修改操作。

考虑动态 dp

每个区间只要维护 4 个值 (左端点是否选，右端点是否选) 就能进行区间的合并
我们可以在线段树上每个区间都维护这 4 个值，就能通过把 $logn$ 个区间合并得到每次询问的答案
同时线段树可以很方便的支持修改

```
f[left][0]表示左半边右端点不选的和最大是多少
f[left][1]表示左半边右端点选的和最大是多少
f[right][0]表示右半边左端点不选的和最大是多少
f[right][1]表示右半边左端点选的和最大是多少
ans=max(f[left][0]+f[right][1],f[left][0]+f[right][0],f[
    left][1]+f[right][0])
```

例题

Description

给出一个大小为 n 的树，每个点有值 a_i，多次询问，每次让你从 x 到 y 的路径上中选出若干个数，选择的数不能相邻，问选的数的和最大是多少。带有修改操作。

考虑动态 dp

与之前一样的思路，不过树上的链问题用线段树无法解决，可以使用树链剖分。

目录

cdq 分治

Description

cdq 分治是基于离线的分治算法当一个问题是不强制在线的，我们可以进行离线操作——对问题的某一维度进行分治
一般来说，CDQ 分治分为两步
1. 递归解决区间 $[l, mid]$ 和 $[mid+1, r]$
2. 统计分治的前段区间的操作对后段区间中查询操作的影响
一般来说，cdq 分治的能用多 $logn$ 的复杂度对问题进行降维

归并排序求逆序对

Description

求逆序对本质上是一个二维偏序问题，一维是位置，一维是数值。
对位置这维分治后，在多一个 log 复杂度的情况下将问题降为只剩数值这维。
我认为这是最简单的 cdq 分治。

例题

Description

有一个 $n \times n$ 的矩阵，所有格子的初始值均为 0。
每次操作可以增加某格子的权值，或询问某子矩阵的总权值。

分析

先通过二维前缀和的转化，将一个询问拆成 4 个对二维前缀和的询问。
本质是一个三维偏序问题，时间、横纵坐标这三维。
可以通过排序去掉横坐标这维，cdq 对时间这维分治，纵坐标那维用树状数组/线段树做前缀查询。

目录

点分治

Description

点分治适合处理大规模的树上路径信息问题。
点分治，顾名思义就是基于树上的节点进行分治，对于点的拆开其实就是对于树的拆开，本质其实是将一棵树拆分成许多棵子树处理，并不断进行。
假设我们选取了当前分治中心 *root*, 所有完全位于 *root* 子树中的路径可以分为两种，一种是经过 *root* 的路径，一种是不经过 *root* 的路径。对于经过 *root* 的路径，又可以分为两种，一种是以 *root* 为一个端点的路径，另一种是两个端点都不为 *root* 的路径。而后者又可以由两条属于前者链合并得到。所以，对于 *root*，我们先计算在其子树中且经过该节点的路径对答案的贡献，再相当于把这个点删除，递归其子树对不经过该节点的路径进行求解。这样，每个点都作为 *root* 计算完成后，整棵树的所有路径也计算完毕。

点分治

Description

怎样选取最合适的分治中心点
如果树退化为一个链
选择链首：递归 n 层，总复杂度 $O(n^2)$(假设计算一个子树的复杂度是线性)
选择链心：递归 $logn$ 层，总复杂度总复杂度 $O(nlogn)$(假设计算一个子树的复杂度是线性)
通过这个例子，我们不难发现：
如果选点后子树越大，递归层数越多，时间越慢，反之则越快。
所以我们的选点标准就是要尽量平衡的分成若干子树。

点分治

Description

重心的定义：
找到一个点，以其为根时，其所有的子树中最大的子树节点数最少，那么这个点就是这棵树的重心。
重心的性质：
以重心为根时，它每个子树大小不会超过整个树大小的一半。
树中所有点到某个点的距离和中，到重心的距离和是最小的，如果有两个重心，他们的距离和一样。
把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。
一棵树添加或者删除一个节点，树的重心最多只移动一条边的位置。
一棵树最多有两个重心，且相邻。

点分治

找重心

```
void getroot(int u,int fa){
  sim[u] = 1; mxson[u] = 0;
  for (int i = head[u];i;i = t[i].next){
      int v = t[i].to;
      if(vis[v]||v == fa) continue;
      getroot(v,u);
      sim[u] = sim[u] + sim[v];
      mxson[u] = max(mxson[u],sim[v]);
    }
  mxson[u] = max(mxson[u],Smer - sim[u]);
  if(mxson[u]<MX){root = u; MX = mxson[u];}
}
```

点分治

Description

因为将从根出发的所有链两两拼接有可能这两条链先在子树中有交，所以要减掉这部分算多的。

```
void Divide(int rt)
{
  ans = ans + solve(root,0); // 求出经过当前根的贡献
  vis[rt] = true;
  for(int i = head[rt];i;i = t[i].next)
    {
      int v = t[i].to;
      if(vis[v])continue;
      ans = ans - solve(v,t[i].len); // 减去多算的
      Smer = sim[v]; root = 0;
      MX = INF;
      getroot(v,0); // 找子树的重心
      Divide(root);
    }
  return;
}
```

例题

Description

给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在。

分析

考虑点分治，统计答案的时候记录从根到子树所有点的链的长度，就可计算答案。

点分树

Description

给定一棵有 n 个点的树，每个点有权值 a_i。
多次操作，每次可以询问距离点 x 不超过 y 的所有点的权值和，也可以修改一个点的权值。
强制在线。

分析

考虑用点分治的每次的分治中心构成一个树形结构，也就是点分树，树高是 $logn$
每次查询时枚举 lca，查询该子树内距离小于 d 的权值和（线段树维护），注意去重。
每次修改时也是枚举 lca，修改线段树。