主席树小结

SGColin

目录

1 一些题目

1.1 [POI2014]KUR-Couriers

1.1.1 Description

Luogu 3567

给一个长度为 n 的序列 $a, 1 \leq a_i \leq n$。多组询问，每次询问一个区间 $[l, r]$，是否存在一个数在 $[l, r]$ 中出现的次数大于 $\frac{(r-l+1)}{2}$。如果存在，输出这个数，否则输出 0。

1.1.2 Solution

直接查区间众数是没有比较优秀的复杂度的，但超过半数的限制就可以保证复杂度了。每次询问显然至多只有一个子节点满足出现次数超过半数，只进那个子节点即可。注意可持久化权值线段树不要维护区间出现次数最大值，因为差分以后可能并不是出现最多的。

1.2 [湖南集训] 谈笑风生

1.2.1 Description

Luogu 3899

给出一棵以 1 为根的树，每次给出一个二元组 (u, k) ，询问三元组 (u, b, c) 的个数：u, b, c 均不同，且 u, b 均为 c 的祖先，u, b 的距离不超过 k。

1.2.2 Solution

将 b 分成 u 的祖先和 u 的子树中的点考虑。设 $d[u]$ 表示 u 的深度，$sz[u]$ 表示 u 子树的大小。

如果 b 为 u 的祖先，则 c 只能在 u 子树里选，这部分的答案是 $\min(k, d[u]) \times (sz[u] - 1)$。

如果 b 为 u 的子树内节点，则应选择 $v \in son[u], d[v] \in [d[u] + 1, d[u] + k]$ 的子节点 v 作为 b ，这部分答案是 $\sum_v (sz[v] - 1)$。

第一部分预处理后是 O(1) 的，第二部分可以对深度开权值线段树，然后区间查值。因为多组询问，所以按照 dfs 序建主席树即可。

1.3 [CQOI 2015] 任务查询系统

1.3.1 Description

BZOJ 3932

有多个任务，被描述为三元组 (l_i, r_i, w_i)，代表其从 l_i 一直被执行到 r_i ，其重要度为 w_i 。现在多次询问某一个时间点，重要度前 k_i 小的重要度之和。

1.3.2 Solution

想要查前 k 小的和用值域线段树就好了。问题在于每个时间点开一个至于线段树，那区间加就变成了区间内每个线段树都插入一个值。

差分后求前缀和的思路可以在这里得以应用。注意到主席树每次继承上一个版本实际上是取了一个前缀和，所以我们可以在建树之前把区间插入给差分掉，然后每次继承前缀的版本即可。具体的，把每次覆盖拆成加入和删除两次修改，然后按时间扫描建树。询问直接在对应的树上查。

1.4 [CTSC 2018] 混合果汁

1.4.1 Description

UOJ 402

有 n 种果汁，i 号的美味度是 d_i，**每升价格为** p_i，在一瓶混合果汁中最多只能添加 l_i 升。

共 m 次询问，混合果汁总价格不大于 g_j，体积不小于 L_j 的限制条件下，混合果汁的美味度尽可能地高。定义混合果汁的美味度等于所有参与混合的果汁的美味度的最小值。

1.4.2 Solution

注意别读错题，是每升价格为 p_i ....

考虑二分答案，那么我们就确定了可选的集合，即 $d_i >$ 二分值的所有果汁。那么取果汁的策略就确定了，每次挑单价最小的，然后把 l_i 升全部用完，然后再用下一个单价最小的，以此类推。

这个过程可以用一个以价格为下标的权值线段树加速。那么考虑把每个二分答案对应的权值线段树建出来，就可以进行二分了。注意到每一个 d_i 对应的集合都包含比他大的 d_i 对应的果汁，所以将 d_i 排序后由大到小建主席树。之后每次二分在对应的主席树上查询即可。

1.5 [FJOI 2016] 神秘数 & [Codechef FRBSUM] ForbiddenSum

1.5.1 Description

BZOJ 4299

数集 S 的 ForbiddenSum 定义为无法用 S 的某个子集（可以为空）的和表示的最小非负整数。

给出一个序列 A ，询问该数列的一些子区间所形成的数集的 ForbiddenSum 。

1.5.2 Solution

如果之前可以组成前 R 个数，那么加入一个 $x \leq R+1$ 的数构可以使上界变成 $R+x$，否则对当前无贡献。结论是正确的。对于 $[1,R]$ 范围内的数之前就保证了可以组合出，而 $[R+1,x]$ 范围内的数直接减掉 x 就可以组合出。

所以做法就很清楚了。假设当前已知的可表出区间为 $[1,R]$ ，那么查询询问区间内所有 $[1,R+1]$ 范围内的数的和 S，如果 $S \geq R$ 则可以继续将区间扩展为 $[1,S]$，否则结束。最差的情况是每个查询的范围内只有一个数，那么每次查询的就是 $1,2,4,8,...,2^k$，可以看出查询次数的级别是 log 的。

支持查一个值域区间的所有元素和，用权值线段树，因为区间询问我们可持久化就好了。

1.6 [IOI 2014] Holiday

1.6.1 Description

UOJ 29

给出一个序列，每个位置有权值 a_i，开始你在第 s 个位置。现在你一共可以行动 t 秒，每秒可以移动到两侧的位置，或取走当前位置的权值，每个位置的权只能被取走一次，位置可以被多次访问，最大化取得的权值和。

1.6.2 Solution

考虑我们确定了要向左走 x 个位置，向右走 y 个位置，那么获得的权值怎么计算。首先肯定有一侧要折回来，所以剩下的时间为 $k = t - x - y - \min(x, y)$ 用来取权值，此时我们显然要取区间 $[s - x, s + y]$ 内的前 k 大，要支持查询区间查前 k 大和，主席树即可。此时我们枚举两侧端点，复杂度为 $\mathrm{O}(n^2 \log n)$。

考虑优化它。我们分开考虑向左走折回来和向右走折回来的情况。

对于向左走折回来的情况，可以发现它选择的右端点是有决策单调性的。设 $p[x]$ 为从开始向右分配 x 秒最终停下的最靠右的最优位置，那么如果存在 $p[x + 1] < p[x]$ ，那么显然 $p[x + 1]$ 对于 x 来说决策点更优秀，$p[x]$ 的定义就不合法了。

因此我们可以分治。设 solve(l,r,L,R) 表示当前要解决的左端点是 $[l, r]$，需要折回来，向右走的可能答案区间为 $[L, R]$ 。那么我们暴力枚举 $L...R$ ，计算出对于左端点在 $mid = \frac{l+r}{2}$ 时的答案和最优转移点 p，接下来分治 solve(l,mid-1,L,p) 和 solve(mid+1,r,p,R) 即可。

类似的处理向右走再回来的情况，所有位置算出的答案取 max 即可。注意需要考虑上只走一侧的情况。

```
void solve1(int l, int r, int L, int R){ //计算向左走再折回来的情况
  if (l > r) return;
  ll res = 0, tmp, ansp = 0;
  for (rg int p = L, rem; p <= R; ++p) {
    if (p == s) rem = m - (s - mid);
    else rem = m - (s - mid) * 2 - (p - s);
    if (rem <= 0) break; //若可选点数小于0则无意义
    tmp = query(rot[mid - 1], rot[p], 1, tot, rem);
    if (tmp > res) { res = tmp; ansp = p;}
  }
  ans = max(ans, res);
  solve1(l, mid-1, L, ansp);
  solve1(mid + 1, r, ansp, R);
}
```

1.7 [UNR #1] 火车管理

1.7.1 Description

UOJ 218

维护排成一排的 n 个栈，要求支持区间压栈一个值，单点弹栈，查区间栈顶的和。

1.7.2 Solution

考虑没有弹栈就显得很简单了。操作就变成了区间赋值，区间求和，通过维护一个栈序列的线段树即可实现。

那么弹栈其实可以看成某一个位置回到了历史版本，考虑主席树。那么每个位置还需要记录上一次出现的位置吗？其实不用。我们把每次压栈的值记录到对应的操作编号上，每个节点记录当前位置所属于的压栈版本即可。那么我们只需要找到弹栈位置对应的压栈版本 x ，然后查一下第 $x-1$ 个版本的主席树该位置是什么版本的压栈即可。

还有一个问题是区间加标记的下放。一个节点的左右子节点都有可能是以前版本的节点，所以下放之后会造成以前版本查询错误。有一个科技就是把左右子节点重建，放弃老节点，申请新节点。注意到这个操作只会在最新版本出现，所以不必担心需要修改后续版本指针的问题。

这样每次操作会涉及 log 个节点（区间），每个区间只会重建儿子。根据线段树区间操作的时间复杂度分析，所以总的空间复杂度还是 $\mathrm{O}(n \log n)$ 的。

实现上有一些细节可以优化，诸如节点无标记就不新建子节点，详见代码。

```
inline void pushdown(int rt, int l, int r) {
  int ls = c[rt].ls, rs = c[rt].rs;
  c[c[rt].ls = newnode()] = c[ls];
  c[c[rt].rs = newnode()] = c[rs];
  c[c[rt].ls].sum = (mid - l + 1) * val[c[rt].id];
  c[c[rt].rs].sum = (r - mid) * val[c[rt].id];
  c[c[rt].ls].id = c[c[rt].rs].id = c[rt].id;
  c[rt].id = 0;
}

if (op == 2) {
  l = (rd() + lstans * ty) % n + 1;
  int id = queryid(rot[i], 1, n, l);
  if (id) update(rot[i], 1, n, l, l, queryid(rot[id - 1], 1, n, l));
}
```