CodeForces Round 525 (Div.2) 解题报告

SGColin

目录

1 A. Ehab and another construction problem

1.1 Description

给出一个整数 n $(n < 100)$ ，求两个整数 a, b $(1 \le a, b \le n)$ 满足：

1. $a \times b > n$

2. $b \mid a$, $\frac{a}{b} < n$

1.2 Solution

签到题。比较愚蠢的方法是直接枚举 a, b，然后判断是否符合要求。

其实再仔细想想，只要 $n > 1$ 的时候，$a = n, b = n$ 一定是一组合法解。

2 B. Ehab and subtraction

2.1 Description

给出一个可重数集，执行以下操作 k 次：

找出数集中最小的正数，输出，并将数集中所有数都减掉它。

如果数集中没有正数则只需输出 0 。

2.2 Solution

签到题。稍微想一想，答案就是排序后序列进行差分，得到的所有正数都输出即可。

3 C. Ehab and a 2-operation task

3.1 Description

给出一个长度为 n 的数列 a，你可以执行一下两种操作：

$1\ i\ x$ 将 $a[1] \sim a[i]$ 都加上 x 。

$2\ i\ x$ 将 $a[1] \sim a[i]$ 都对 x 取模。

要求输出一种构造方案，使得总操作次数不超过 $n+1$ 次，并且操作后的序列为严格递增。

3.2 Solution

比较有意思的构造题目。有两种解法。

要注意到前缀操作的局限性，也就是只加不会改变前大后小的关系。

执行 n 次加法和 1 次取模

当时想出来的做法，看到题目限制联想到最后 1 次操作是全局取模。

注意到取模是可以将整个数列的前缀控制在 x 剩余系下的，因此可以在最后对 $n+1$ 取个模。

那么我们的任务就是把数列变成模意义下 $a[i]=i$ 的数列。这个操作就很简单了，正反扫描均可，只需注意前缀操作对一个位置前面是有影响的。

```
for(int i=n,s=0,tmp;i;--i){
  a[i]=(a[i]+s)%(n+1);
  (s+=(tmp=(n+1+i-a[i])%(n+1)))%=n+1;
  printf("1␣%d␣%d\n",i,tmp);
}
printf("2␣%d␣%d\n",n,n+1);
```

执行 1 次加法和 n 次取模

官方题解给出来了一种更巧妙的方法。同样将数列构造成 $a[i]=i$ 的形式。

首先因为取模只能得到减法的效果，因此为了避免 $a[i]<i$ 的情况，我们先进行一次 $1\ \infty\ n$ 操作即可避免这一问题 [1]。

然后只需让 $a[i]$ 通过模之后减小到 i 了，最直接的方法选择的模数就是 $a[i]+\infty-i$ ，相当于直接减掉多余部分。这样选择的优秀之处在于，它不会因为是前缀操作而对前面的元素有影响，因为前面的数取模后显然 $<a[i]+\infty-i$。

[1] 这里的 ∞ 并不是真的无穷，设为一个给出值域里的最大值即可

4 D. Ehab and another another xor problem

4.1 Description

现在有两个数 a, b $(a, b \leq 2^{30})$，你可以至多进行 62 次输入两个数 c d 交互：

若 $a \oplus c > b \oplus d$，返回 1。

若 $a \oplus c = b \oplus d$，返回 0。

若 $a \oplus c < b \oplus d$，返回 -1。

其中 $\oplus$ 代表异或 (xor) 运算。最后输出 a, b 的值。

4.2 Solution

人生第一道交互。思维难度个人认为不低。

注意到 62 与 30 的关系，很容易想到按二进制位处理。对每一位会有两次询问。对于较低位的询问，答案会受到较高位的影响，因此数字要从高到低确定，进而可以通过异或把影响去掉。

首先我们考虑 a, b 的二进制最高位 a_1, b_1 如何确定（这里看作 30 位）。

1. 如果 $a_1 < b_1$，则询问 2^{30} 0 或 0 2^{30} 的答案都与原数去掉这一位后的大小关系相同。

2. 如果 $a_1 > b_1$，则询问 2^{30} 0 或 0 2^{30} 的答案都与原数去掉这一位后的大小关系相同。

3. 如果 $a_1 = b_1$，且 $a_1 = b_1 = 0$，则询问 2^{30} 0 应该是 1，询问 0 2^{30} 应该是 -1。

4. 如果 $a_1 = b_1$，且 $a_1 = b_1 = 1$，则询问 2^{30} 0 应该是 -1，询问 0 2^{30} 应该是 1。

显然情况 $3, 4$ 比较好判断出，因此现在我们的问题就在于如何区分 $1, 2$。

最基础的想法，是直接输入 0 0，然后即可得出大小关系，因为二进制的特殊性，进而可以知道哪一个是 1。但是问题在于，交互次数上限是 62 次，而每一位询问两次就用掉了 60 次，根本不够后面用的。但是我们可以在一开始确定两数的大小关系，这只消耗一次交互次数。

然后考虑大小关系什么时候可以得到，其实就是上一次出现情况 $1, 2$ 时。因为上一次出现 $1, 2$ 时的答案，就是把上一个 $1, 2$ 情况异或掉后两数大小的比较，且这一位置到上一位置之间的数显然相同，因此当前大小关系就是当时的答案。

```
int a=0,b=0,mx=query(0,0);
for(R int i=29,tmp,ans1,ans2;~i;--i){
  tmp=(1<<i),ans1=query(a+tmp,b),ans2=query(a,b+tmp);
  if(ans1!=ans2){if(ans1==-1) a+=tmp,b+=tmp;}
  else{(mx==1)?a+=tmp:b+=tmp;mx=ans1;}
}
printf("!␣%d␣%d\n",a,b);
```

5 E. Ehab and a component choosing problem

5.1 Description

给出一棵树，点有点权。定义连通块为一个点集，且满足点集中的任意两点在树上的简单路径不会经过点集以外的点。定义连通块的权值为，其包含的所有点的点权之和。

现在需要你找出一些连通块，满足：

1. 任意的两个连通块所对应的的点集之间无交。

2. 这些连通块的权值平均值最大。

3. 当满足上述两条时，应找出尽可能多的连通块。

输出所选的连通块平均值和个数即可。

5.2 Solution

审题。可以发现平均值最大这一限制非常奇怪，当所有连通块权值都不同时，显然只选权值最大的，这样才能最大化平均值。

因此平均值的答案是固定的，一遍树形 DP 即可。转移很好想，当子树贡献答案 >0 的时候就选择接上这棵子树，否则不选。

然后考虑个数的问题模型，实际上就是在树上选出权值为 ans 的连通块，最多能选出不交的几个。这个过程我们可以通过树形贪心得到。转移的过程与上一问相同，如果当前节点所统计出的连通块权值为 ans，就累加计数器，同时当前节点不向父节点贡献权值即可。关于贪心正确性，可以理解为二选一问题上不存在优劣关系，此时最小化节点数显然是更优秀的选择。

```
ll dfs1(ll u,ll fa){
  ll sum=val[u];
  for(ll i=hd[u],v;i;i=e[i].nxt)
    if((v=e[i].to)!=fa) sum+=max(0ll,dfs1(v,u));
  ans=max(sum,ans);
  return sum;
}
ll dfs2(ll u,ll fa){
  ll sum=val[u];
  for(ll i=hd[u],v;i;i=e[i].nxt)
    if((v=e[i].to)!=fa) sum+=max(0ll,dfs2(v,u));
  if(sum==ans) ++cnt,sum=0;
  return sum;
}
```

6 F. Ehab and a weird weight formula

6.1 Description

给出一棵 n ($n \leq 5 \times 10^5$) 个节点的树，点有点权。保证最小权值点只有一个，其他点都至少由一个相邻的点权值比它小。现在要你用原树的节点重构这棵树，最小化新树的代价。

一个点 u 累积的代价是，$val_u \times deg_u$ ，其中 deg_u 为这个点在新树中的点度。

一条新树中的边 $\{u,v\}$ 的代价是，$\min(val_u, val_v) \times \lceil log_2(dist(u,v)) \rceil$ ，其中 $dist(u,v)$ 表示的是原树中两点 u,v 的简单路径长度。

6.2 Solution

最后还是看了题解，感觉此题也比较神了。官方题解讲的是重新考虑边权的定义方式，将点权的代价加入到边权里。也就是说新添加一条边，会增加边权与连接两点的点权。

考虑按照点权从小到大将点加入到树里，对于新加入的点 i ，满足 $val_i > val_j$ ，$j \in tree$。那么新生成一条边，带来的总代价就是 $val_j \times (1 + \lceil log_2(dist(u,v)) \rceil) + val_i$。

首先我们证明一个引理：如果以权值最小的节点作为根去看待原树，那么整棵树一定满足叶节点权值 $>$ 父节点权值，也就是说，在一条由根出发的链上，深度越深的点一定权值越大。其证明通过反证法得到：如果存在一条链上出现小-大-小的权值分布，那么深度较深的点必然需要一个更深的点权值比其小，那么到链底就无法再找到节点比其小了，因为权值最小点为根节点。

注意到这类似一棵 MST 的生成过程。我们以权值最小的节点作为根去看待原树，那么一个点连接的点只会是在原树中它到根路径上的节点。证明很简单，如果它连接了一条边到这条链以外，那么得到的 $dist$ 不会优于根节点的答案，并且根据根节点的定义 val 也必定不优于根节点。

同时进一步的，其实我们可能选择的节点只会在根节点和自底向上距离 2 的整次幂的位置上。因为其他位置一定与一个深度较低的特殊点求得的 $dist$ 的对数相同，但是其权值通过上面的证明一定比特殊点大，但特殊的，没有节点在权值上优于根节点。

因此我们只需以权值最小的节点作为根去看待原树，然后对每个节点构造倍增数组，然后只需在这些倍增数组指向的点里选择建边代价最小的一个即可。

```
void dfs(ll u,ll fa){
  ll ans=val[fa]+val[u];
  for(R ll i=1;i<=t;++i){
    f[u][i]=f[f[u][i-1]][i-1];
    if(f[u][i]!=0) ans=min(ans,val[f[u][i]]*(i+1)+val[u]);
    else ans=min(ans,val[m]*(i+1)+val[u]);
  }
  if(u!=m) res+=ans;
  for(R ll i=hd[u],v;i;i=e[i].nxt) if((v=e[i].to)!=fa){f[v][0]=fa;dfs(v,u);}
}
```