B.I.L. 签到题

A.Swapity Swapity Swap

题意:

一个数组，初始状态为1,2,...,n， 每一回合将会对数组中的一些区间进行反转操作，求K回合操作后的数组

题解:

可以先处理出一回合后原来在位置i的数被转移到了什么位置，记位$p[i]$，可以$O(NM)$暴力求出来。对于每个点，可以发现他没经过某个时间x就会回到原来的位置，即他的路径是一个长度为x的环。而K回合后它的位置就是$K\%x$回合后的位置。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll; typedef double db;
typedef pair<int, int> pii; typedef pair<ll, ll> pll;
typedef pair<int, ll> pil; typedef pair<ll, int> pli;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 1e6 + 50;
int p[MAXN];
bool vis[MAXN];
int val[MAXN];
int ans[MAXN];
void work()
{
    int N,M,K;
    scanf("%d%d%d",&N,&M,&K);
    for(int i=1;i<=N;i++)p[i]=i;
    for(int i=1;i<=M;i++)
    {
        int u,v;scanf("%d%d",&u,&v);
        for(int j=u,k=v;j<k;j++,k--)
        {
            swap(p[j],p[k]);
        }
    }
    for(int i=1;i<=N;i++)
    {
        if(vis[i])continue;
        int now=i,nxt=p[now];
        int cntH=0;val[cntH++]=now;vis[now]=1;
        while(!vis[nxt])
        {
            now=nxt;vis[now]=1;nxt=p[now];
            val[cntH++]=now;
        }
        int QAQ=K%cntH;
```

```
        for(int j=0;j<cntH;j++)
        {
            ans[val[j]]=val[(j+QAQ)%cntH];
        }
    }
    for(int i=1;i<=N;i++)printf("%d\n",ans[i]);
}
int main(){
    work();
}
```

--ztc

C.Delegation 1

自己画一个图，想一下`rest`有哪些情况需要转移，`cnt`如何计数。

流程就是枚举K，然后dfs，回溯的时候处理合并，想通了怎么合并这题也没什么东西了。

英文注释是我赛内写的，不翻译了。

```
using pii = pair<int, int>;
using vint = vector<int>;

int rst[MAXN];  // rest edge in subtree
int cnt[MAXN];  // count for possible path in subtree
int n, m, amount, len;
vint g[MAXN];
int ans[MAXN];
bool valid;

// map<int, int> match; 如果你有其他优化的话，可以尝试map，很省代码
int match[MAXN], match_cnt;
int stk[MAXN], stk_top;

void dfs(int u, int f) {
    if (!valid) return;
    rst[u] = 0;
    cnt[u] = 0;
    for (auto v:g[u]) {
        if (v == f)continue;
        dfs(v, u);
        cnt[u] += cnt[v];
    }
    if (!valid) return;
    for (auto v:g[u]) {
        if (v == f)continue;
        if (rst[v] + 1 == len) {
            cnt[u]++;
        } else {
            int rst_need = len - rst[v] - 1;
            if (match[rst_need] > 0) {
                match[rst_need]--;
                match_cnt--;
                cnt[u]++;
```

```
            } else {
                match[rst[v] + 1]++;
                match_cnt++;
                stk[stk_top++] = rst[v] + 1;
            }
        }
    }
    // 这一段都在优化最后剩下的是不是一个无法匹配的rest，  避免一个map的复杂度。
    int final_rst = -1;
    while (stk_top > 0) {
        int top_item = stk[--stk_top];
        if (match[top_item] > 0) {
            final_rst = top_item;
            match[top_item] = 0;
        }
    }
    if (match_cnt > 1) {
        valid = false;
        return;
    } else if (match_cnt == 1) {
        rst[u] = final_rst;
        match_cnt = 0;
    }
        // 到这里结束
    return;
}
// fast input
inline void read(int &x) {
    char ch;
    bool flag = false;
    for (ch = getchar(); !isdigit(ch); ch = getchar())if (ch == '-') flag =
true;
    for (x = 0; isdigit(ch); x = x * 10 + ch - '0', ch = getchar());
    x = flag ? -x : x;
}

void solve(int kaseId = -1) {

    read(n);
    m = n - 1;
    for (int i = 1, u, v; i < n; ++i) {
        read(u);
        read(v);
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }

    // amount is amount of roads.
    for (len = 1; len <= m; ++len) {
        if (m % len != 0) continue;
        amount = m / len;
        valid = true;
        match_cnt = 0;
        stk_top = 0;

        dfs(1, -1);
        if (valid && rst[1] == 0 && cnt[1] == amount) {
            ans[len] = 1;
```

```
            }
        }
        for (int i = 1; i <= m; ++i) {
            printf("%d", ans[i]);
        }
        puts("");
    }
```

--wtw

D.Triangles 2

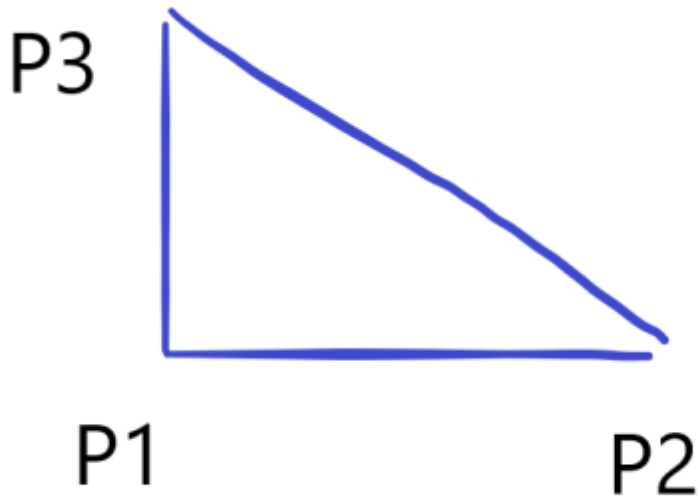
题意

二维平面上有1e5个点，输出所有可以组成的直角边平行坐标轴的直角三角形的面积的和。

做法

傻瓜做法，我们将所有的三角形分成四个图案，

对于每一种类型的三角形，我们保证直角顶点会在最后每枚举到，这个顺序可以用不同条件下的sort来保证。



开始做题的时候，对于某一类型的直角三角形（以上面第一个为例子），这样的情况下我们按照**先x从大到小，后y从大到小**的顺序进行sort。

那么我们在for的过程中，一定会先碰到P2点，这个时候还没有答案，我们将这个点的影响维护起来，

```
sx[P2.x]+=P2.y,sy[P2.y]+=P2.x
```

即在之后如果还有碰到y为P2.y的点，那么这个点就可以被作为上面三角形的右边的顶点，如果还有碰到x为P2.x的点，那么这个点就可以被作为上面三角形的上面的顶点。

当然还要维护这个y有多少个顶点，即下面代码中的nx和ny。

那么最后我们for到P1点时，可以知道上面的P3和P2在维护后使得P1.x和P1.y的数量不为0，即有值，那么我们就可以直接减掉后abs处理啦。

有问题欢迎私聊，希望写的题解有人看..

代码

```
#include<bits/stdc++.h>
#define fi first
#define se second
#define rep(i,a,b) for(int i=(int)a;i<=(int)b;i++)
#define pb push_back
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long ll;
typedef pair<int,int> pii;
const int maxn=100005;
const int maxm=20005;

ll mod=(ll)1e9+7,ans;
ll sx[maxm],sy[maxm];
ll nx[maxm],ny[maxm];
int n;
struct node{
    ll x,y;
}e[maxn];
bool cmp1(node a,node b){
    if(a.x==b.x) return a.y<b.y;
    return a.x<b.x;
}
bool cmp2(node a,node b){
    if(a.x==b.x) return a.y<b.y;
    return a.x>b.x;
}
bool cmp3(node a,node b){
    if(a.x==b.x) return a.y>b.y;
    return a.x<b.x;
}
bool cmp4(node a,node b){
    if(a.x==b.x) return a.y>b.y;
    return a.x>b.x;
}
void init(){
    memset(nx,0,sizeof(nx));
    memset(ny,0,sizeof(ny));
    memset(sx,0,sizeof(sx));
    memset(sy,0,sizeof(sy));
}
void deal(){
    rep(i,1,n){
```

```
        ll x=e[i].x,y=e[i].y;
        if(nx[x]&&ny[y]){
            ans=(ans+abs((x*ny[y]-sy[y])*(y*nx[x]-sx[x]))%mod)%mod;
        }
        nx[x]++,ny[y]++;
        sx[x]+=y,sy[y]+=x;
    }
}
int main(){
    scanf("%d",&n);
    rep(i,1,n) scanf("%lld%lld",&e[i].x,&e[i].y),e[i].x+=10000,e[i].y+=10000;
    sort(e+1,e+1+n,cmp1);
    deal();

    sort(e+1,e+1+n,cmp2);
    init();
    deal();

    sort(e+1,e+1+n,cmp3);
    init();
    deal();

    sort(e+1,e+1+n,cmp4);
    init();
    deal();
    printf("%lld\n",ans);
    return 0;
}
```

--yh

E： Delegation2

大概题意：一棵树，把它分成一些链，使得最短的链最长。

Solution： 二分答案，进行check，让每一个节点向上传递一根最长的链

Check的时候维护每个点的子链长度，当check到叶子节点的时候把tmp数组的值设成1。在check其它节点的时候，先for一遍子结点，当扫到标记为-1的子结点（意思是这个子结点及其孙子节点无法向上提供链）时向multiset里面插入1，其它情况只要向multiset里面插入tmp[子结点标号]就行。插入完成之后可以考虑对子链进行合并，注意这里每条合并之后的链最多由两条初始子链合并成。由于我们要考虑尽可能使当前节点能够提供上方节点可用的链长度最长，所以合并的时候可以考虑每次把set的begin拿出来，然后进行lower_bound(mid-*st.begin())，搜到的时候要考虑到第一个元素是2，mid=4这种情况，这样他会搜索到自身，所以判断一下。

这题还有一个注意点就是当一个节点的子链个数是偶数个并且子链的最大长度大于等于mid，就要删去一条子链中的最长链。考虑某个节点的子链有下面这种情况

mid：8　　子链长度：1 2 5 7 10 10

这时侯直接合并就会变成 1+7 2+10 5+10，导致没有其它链返回给上一层使用。所以这时候就要删掉一根长度为10的链，让他单独成链，这时候合并就会变成1+7 2+10 5，剩下一根长度为5<mid的链，那么就要把这条链返回给上一层使用。

复杂度分析：二分+dfs = nlogn 由于每一次check时每个节点只会进入multiset一次，出去一次，lower_bound还有个log 所以 二分+dfs+multimap 差不多 nlognlogn(？)

下面是改了N遍的代码，感觉测试数据有不完善的地方，第一次比赛时AC的代码被自己用个小样例就hack掉了

```
#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
using namespace std;
#define mst(a,b) memset(a,b,sizeof(a))
#define ll long long
#define ull unsigned long long
#define fi first
#define se second
typedef pair<int, int>pii;
typedef pair<ll, int>pli;
typedef pair<int, ll>pil;
typedef pair<ll, ll>pll;
const ull mo = (ull)1e9 + 7;
ll gcd(ll a, ll b) { return b == 0 ? a : gcd(b, a % b); }
int n;
vector<int> v[110000];
vector<int> hase[110000];
int tmp[110000];
ll mid;
int check(int cur, int pre)
{
    ///叶子节点设值后直接返回
    if (v[cur].size() == 1 && v[cur][0] == pre)
    {
        tmp[cur] = 1;
        return 0;
    }
    multiset<int> st;
    for (int i = 0; i < v[cur].size(); i++)
    {
        if (v[cur][i] != pre)
        {
            ///tag为-1代表已经出错
            int tag = check(v[cur][i], cur);
            if (tag == -1)
            {
                return -1;
            }
            else if (tmp[v[cur][i]] != -1)
            {
                st.insert(tmp[v[cur][i]]);
            }
            else
            {
                st.insert(1);
            }
        }
    }
    ///如果子链个数为偶数且最长子链长度大于等于mid就删掉最长子链
    auto q = st.end();
    q--;
    if (st.size() > 0 && st.size() % 2 == 0 && *q >= mid)
    {
        st.erase(q);
```

```
    }
    ///给当前节点赋初值
    tmp[cur] = -1;
    ///开始合并
    while (!st.empty())
    {
        ///合并时如果当前最短链都>=mid，那么肯定就可以供一个长度为mid的链给上一级使用
        if (*st.begin() >= mid)
        {
            tmp[cur] = mid + 1;
            return 0;
        }
        if (st.size() == 1)
        {
            ///当前合并到只剩一个元素并且该元素长度小于mid
            ///如果当前还没有选向上传递的链，那么就把这一条链长度+自己到上个节点的长度(也就是
1)传上去
            ///如果已经有了向上传的链，那么要传两条就肯定错了，直接返回-1
            if (tmp[cur] == -1)
            {
                tmp[cur] = *st.begin() + 1;
                return 0;
            }
            else
            {
                return -1;
            }
        }
        ///贪心查找使得*p+*begin()>=mid的最小数
        auto p = st.lower_bound(mid - *st.begin());
        if (p == st.begin()) p++;
        ///没找到并且已经选好要向上传递的链了那么就返回-1
        if (p == st.end() && tmp[cur] != -1)
        {
            return -1;
        }
        ///找到了，把两条链消去
        else if (p != st.end())
        {
            st.erase(p);
            st.erase(st.begin());
        }
        ///没找到，选为向上传递的链
        else if (tmp[cur] == -1)
        {
            tmp[cur] = *st.begin() + 1;
            st.erase(st.begin());
        }
    }
    return 0;
}
void work()
{
    cin >> n;
    int x, y;
    for (int i = 1; i < n; i++)
    {
        cin >> x >> y;
```

```
        v[x].push_back(y);
        v[y].push_back(x);
    }
    ll l = 1, r = 1e5 + 100, ans;
    int tag;
    while (l <= r)
    {
        mid = (l + r) >> 1;
        tag = check(1, -1);
        if (tag == 0)
        {
            ans = mid;
            l = mid + 1;
        }
        else
        {
            r = mid - 1;
        }
    }
    cout << ans << "\n";
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    ///int T;cin>>T;while(T--)
    work();
    return 0;
}
/*
附hack小样例:
7
1 2
1 3
2 4
2 5
4 6
5 7
ans:2

8
1 2
1 3
2 4
2 5
5 6
6 7
7 8
ans:3
*/
```

--zxc

F.Timeline

题意:

给了n个事件最早发生的时间，以及m个限制，b_i必须在a_i发生后c_i天之后发生，求每个事件最早发生的时间

题解:

令$dp[u]$表示u最早发生的时间，对于每个限制，就要转移$dp[b_i]=\max(dp[b_i],dp[a_i]+c_i)$,但是在转移之前需要保证$dp[a_i]$已经是最小的了，也就是已经满足了所有对$a_i$的限制，于是根据拓扑序进行$dp$即可

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll; typedef double db;
typedef pair<int, int> pii; typedef pair<ll, ll> pll;
typedef pair<int, ll> pil; typedef pair<ll, int> pli;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN =2e5 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1e9+7;
vector<pii>to[MAXN];
int in[MAXN];
int val[MAXN];
void work()
{
    int n,m,c;scanf("%d%d%d",&n,&m,&c);
    for(int i=1;i<=n;i++)scanf("%d",val+i);
    for(int i=1;i<=c;i++)
    {
        int a,b,x;scanf("%d%d%d",&a,&b,&x);
        to[a].push_back({b,x});
        in[b]++;
    }
    queue<int>q;
    for(int i=1;i<=n;i++)
    {
        if(in[i]==0)q.push(i);
    }
    while(!q.empty())
    {
        int u=q.front();q.pop();
        for(auto v:to[u])
        {
            val[v.first]=max(val[v.first],val[u]+v.second);
            in[v.first]--;
            if(in[v.first]==0)q.push(v.first);
        }
    }
    for(int i=1;i<=n;i++)printf("%d\n",val[i]);
}
int main(){
    work();
}
```

--ztc

G. ClockTree

题意:

一个树，每个点有个点权a_i，每次可以从当前所在的点进入相邻的某个点i，他的点权就变为$a_i = (a_i + 1)$,求有多少个点能够作为起点使得最后所有的点的点权成为12的倍数

题解:

注意到树是个二分图，进行黑白染色，每走两步，当前点的颜色不会发生变化，并且所有的黑点的点权+1，白点也+1，那么如果初始状态下他们之间的差为x，在移动过程中，他们之间的差只可能为(x，x+1)或(x，x-1)（取决于起点是黑点还是白点），如果要使得所有的点都是12的倍数，显然他们之间的差（所有黑点的权值-所有白点的权值）只能是12的倍数，实际上只要存在某一时刻他们的差是12的倍数，就一定存在方案可以满足题目中的条件，于是根据实际情况ifelse即可。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll; typedef double db;
typedef pair<int, int> pii; typedef pair<ll, ll> pll;
typedef pair<int, ll> pil; typedef pair<ll, int> pli;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 3e3 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1e9+7;
int val[MAXN];
vector<int>to[MAXN];
int cntb,cntval;
void dfs(int u,int fa,int col)
{
    if(col)cntb++,cntval+=val[u];
    else cntval-=val[u];
    for(auto v:to[u])
    {
        if(v==fa)continue;
        dfs(v,u,col^1);
    }
}
void work()
{
    int n;scanf("%d",&n);
    bool ok=1;
    for(int i=1;i<=n;i++)
    scanf("%d",val+i),val[i]=12-val[i],ok=(val[i]==0?ok:0);
    for(int i=1;i<n;i++){
        int u,v;scanf("%d%d",&u,&v);
        to[u].push_back(v);
        to[v].push_back(u);
    }
    dfs(1,-1,1);
    cntval=(cntval%12+12)%12;
    if(ok||cntval==0)printf("%d\n",n);
    else if(cntval==11)printf("%d\n",cntb);
    else if(cntval==1) printf("%d\n",n-cntb);
    else printf("0\n");
}
int main(){
```

```
    work();
}
```

--ztc

H: Help Yourself 1

题意

在一维数轴上有n个区间，定义一个区间集合的复杂度为这个区间集合中有多少个并，一个并可以理解为一串区间因为互相之间有重叠部分合并成一个大的区间了，这个大的区间就是一个并。要求这n个区间的所有子集的复杂度之和

思路

由于总共2^n个子集，所以肯定是不能暴力枚举的。考虑单独考虑单个的区间，这个区间会对复杂度的大小产生贡献的情况只有这个区间可以产生一个独立的并的时候，而且一个区间在一个子集里面最多只会产生一次贡献，那么现在问题就转化成了统计某一个区间能在几个区间子集里产生了贡献。

为方便统计，我们统一将某一个并的贡献视为由其中最左边的区间产生。即，在我们将区间按l排序之后，每一个并的贡献就是由我们遍历到的第一个属于这个并的区间产生。

因此，当我们按顺序遍历排序过后的区间，我们只需要考虑这个区间有没有和已经遍历过的区间重合，而不需要考虑那些没遍历过的区间，因为那些没遍历到的区间即使和现在的区间重合也不会影响当前区间产生贡献。由于我们已经按l排过序，所以遍历过的区间会和当前区间重合的充要条件就是$r_j >= l_i, j < i$，这里的话有很多方法可以维护出现过的r里比某个数大的有多少个，我是直接优先队列维护了一下，写起来比较方便，答案的话就是$\sum_i^n 2^{\text{前面有几个和当前区间不重合的区间}} * 2^{\text{后面有几个区间}}$码如下：

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i,j,k) for(int i = (int)j;i <= (int)k;i ++)
#define debug(x) cerr<<#x<<":"<<x<<endl
#define pb push_back

typedef long long ll;
typedef pair<int,int> pi;
const int MAXN = (int)1e6+7;
const ll mod=(ll)1e9+7;
struct node{
    int x,y;
    bool operator<(const node &a)const{
        return x==a.x?y<a.y:x<a.x;
    }
}p[100009];
priority_queue<int, vector<int>, greater<int> > qu;
ll quick(ll x){
    if(x==0)return 1;
    if(x==1)return 2;
    ll y=quick(x>>1);
    y=y*y%mod;
    if(x&1)y=y*2%mod;
    return y;
}
int n;
int main(){
```

```
    scanf("%d",&n);
    for(int i=1;i<=n;i++)scanf("%d%d",&p[i].x,&p[i].y);
    sort(p+1,p+1+n);
    ll ans=0;
    for(int i=1;i<=n;i++){
        while(!qu.empty()&&qu.top()<=p[i].x)qu.pop();
        ans=(ans+quick(i-1-qu.size())*quick(n-i)%mod)%mod;
        qu.push(p[i].y);
    }
    printf("%lld\n",ans);
    return 0;
}
```

--hzj