2020牛客暑期多校训练营（第二场）

比赛情况

题号	A	B	C	D	E	F	G	H	I	J	K
状态	Ø	Ø	O	O	Ø	O	Ø	Ø	Ø	Ø	Ø

O 在比赛中通过 Ø 赛后通过 ! 尝试了但是失败了 - 没有尝试

比赛时间

2020-07-13 12:00-17:00

题解

A - All with Pairs

给出n个字符串，定义$f(s,t)$为s的前缀与t的后缀相等的最大长度。统计$\sum\limits _{i=1}^n\sum\limits _{j=1}^nf(s_i,s_j)$

可以先对所有串的所有后缀哈希，则对于每个串，可以得到第i位前缀与$\text{cnt}[i]$个后缀相同，但这个前缀未必是长度最大的kmp一下$\text{cnt}[\text{nxt}[i]]-=\text{cnt}[i]$即可得到每一位的真正贡献。

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int N=1e5+5;
const int M=1e6+5;
const ll mod=998244353;
string s[N];
int nxt[M],cnt[N];
char p[M];
map<ll,int>mp;
ll res,pw[M];
int main()
{
    int n;
    scanf("%d",&n);
    pw[0]=1;for(int i=1;i<M;i++)pw[i]=pw[i-1]*113;
    for(int i=1;i<=n;i++)
    {
        cin>>s[i];
        ll hash=0,m=s[i].length();
        for(int j=1;j<=m;j++,hash*=113)
```

```
        hash+=s[i][m-j]-'a'+1,mp[hash]++;
    }
    for(int k=1;k<=n;k++)
    {
        int m=s[k].length();
        s[k].copy(p+1,m,0),p[m+1]='\0';
        ll hash=p[1]-'a'+1;cnt[1]=mp[hash];
        for(int i=2,j=0;i<=m;i++)
        {
            while(j&&p[i]!=p[j+1])j=nxt[j];
            if(p[i]==p[j+1])j++;
            hash+=pw[i-1]*(p[i]-'a'+1),cnt[i]=mp[hash];
            nxt[i]=j;
            if(j)cnt[j]-=cnt[i];
        }
        for(int i=1;i<=m;i++)res=(res+1ll*cnt[i]*i%mod*i%mod)%mod;
    }
    printf("%lld\n",res);
    return 0;
}
```

B - Boundary

问过原点的圆周里过给定点数量的最大值。

$n\le 2000$□想到选各点作为圆上一点P则圆心在OP的中垂线上，对于中垂线两两求交点看重合次数□map查询$O(n^2\log n)$

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int N=2e3+10;
const double eps=1e-5;
const double INF=(1LL<<30);
inline ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}
inline ll lcm(ll a,ll b){return a*b/gcd(a,b);}
int dcmp(double x,double y)
{
    double t=fabs(x-y);
    if(t<-eps)return -1;
    return t>eps;
}
struct Point
{
```

```
    double x,y;
    Point(double x=0,double y=0):x(x),y(y){}
    bool operator < (const Point &a) const {return
dcmp(x,a.x)==0?(y<a.y):(x<a.x);}
    bool operator == (const Point &a) const {return
dcmp(x,a.x)==0&&dcmp(y,a.y)==0;}
}p[N];
struct Line
{
    double a,b,c;
    Line(double a=0,double b=0,double c=0):a(a),b(b),c(c){}
}l[N];
inline Point middle(Point p1,Point p2){return
Point((p1.x+p2.x)*0.5,(p1.y+p2.y)*0.5);}
Point ori=Point(0,0);
inline Point intersection(Line l1,Line l2)
{
    if(dcmp(l1.b*l2.a,l2.b*l1.a)==0)return ori;
    double y=(l2.c*l1.a-l1.c*l2.a)/(l1.b*l2.a-l2.b*l1.a);
    double x=(l1.c*l2.b-l2.c*l1.b)/(l1.b*l2.a-l2.b*l1.a);
    return Point(x,y);
}
map<Point,int>mp;
inline ll read()
{
    ll x=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
    return x*f;
}
int main()
{
    int n=read(),res=0;
    for(int i=1;i<=n;i++)
    {
        p[i].x=read(),p[i].y=read();
        Point mid=middle(p[i],ori);
        l[i].a=p[i].x,l[i].b=p[i].y;
        l[i].c=-l[i].a*mid.x-l[i].b*mid.y;
        mp.clear();
        for(int j=1;j<i;j++)
        {
            Point inter=intersection(l[i],l[j]);
            if(inter==ori)continue;
            res=max(res,++mp[inter]);
        }
    }
    printf("%d\n",res+1);
    return 0;
}
```

C - Cover the Tree

题意：最少链覆盖树，输出方案。

首先毫无疑问的是最少链数就是度数为1的节点数目加一除以二，方案的话，对于x的子树中度数为1的节点，如果目前未匹配的有奇数个，那么他们之间可以两两配对，然后留一个伸出去，从而覆盖x到其父亲的边，偶数个则要伸出去两个，这样贪心的输出方案即可。

```
#include <stdio.h>
#include <queue>
#include <string.h>
#include <stdlib.h>
#include <algorithm>
using namespace std;
const int N = 2e5+5;
int head[N],tot;
int du[N];
struct E
{int nxt,to;}e[N<<1];
void add(int x,int y) {
    e[++tot].nxt = head[x];head[x] = tot;e[tot].to = y;
    e[++tot].nxt = head[y];head[y] = tot;e[tot].to = x;
}
queue<int>sons[N];
int anss[N][2],anscnt;
void dfs(int x,int fa) {
    bool isson = true;
    int tomatch = 0;
    for (int i = head[x];i;i=e[i].nxt)
        if (e[i].to!=fa) {
            dfs(e[i].to,x);
            isson = false;
            tomatch += sons[e[i].to].size();
        }
    if (isson) {
        sons[x].push(x);
    }
    queue<int>son2,son1;
    for (int i = head[x];i;i=e[i].nxt)
        if (e[i].to!=fa) {
            if (sons[e[i].to].size()==2)
                son2.push(e[i].to);
            else if (sons[e[i].to].size()==1)
                son1.push(e[i].to);
        }
    if (son2.size()%2==1 && son1.size() == 0 && son2.size()!=1) {
        int x1 = son2.front();son2.pop();
        int x2 = son2.front();son2.pop();
        int x3 = son2.front();son2.pop();
```

```
        anss[anscnt+1][0] = sons[x1].front();sons[x1].pop();
        anss[anscnt+1][1] = sons[x2].front();sons[x2].pop();
        anss[anscnt+2][0] = sons[x1].front();sons[x1].pop();
        anss[anscnt+2][1] = sons[x3].front();sons[x3].pop();
        son1.push(x2);
        son1.push(x3);
        anscnt+=2;
    }
    while (son2.size()>=2) {
        int x1 = son2.front();son2.pop();
        int x2 = son2.front();son2.pop();
        anss[anscnt+1][0] = sons[x1].front();sons[x1].pop();
        anss[anscnt+1][1] = sons[x2].front();sons[x2].pop();
        //anss[anscnt+2][0] = sons[x1].front();sons[x1].pop();
        //anss[anscnt+2][1] = sons[x2].front();sons[x2].pop();
        son1.push(x1);
        son1.push(x2);
        anscnt++;
    }
    while (son2.size()*2 + son1.size() > 2) {
        if (son2.size()) {
            int x1 = son2.front();son2.pop();
            int x2 = son1.front();son1.pop();
            anss[anscnt+1][0] = sons[x1].front();sons[x1].pop();
            anss[anscnt+1][1] = sons[x2].front();sons[x2].pop();
            son1.push(x1);
            anscnt++;
        } else {
            int x1 = son1.front();son1.pop();
            int x2 = son1.front();son1.pop();
            anss[anscnt+1][0] = sons[x1].front();sons[x1].pop();
            anss[anscnt+1][1] = sons[x2].front();sons[x2].pop();
            anscnt++;
        }
    }
    for (int i = head[x];i;i=e[i].nxt)
        if (e[i].to!=fa) {
            while (sons[e[i].to].size())
            {
                int tmp = sons[e[i].to].front();
                sons[x].push(tmp);
                sons[e[i].to].pop();
            }
        }
}
int main()
{
    int n;
    scanf("%d",&n);
    //printf("%d\n",n);
    int x,y;
```

```
    for (int i = 1;i< n;i++)
    {
        scanf("%d%d",&x,&y);
        //printf("%d %d\n",x,y);
        add(x,y);
        du[x]++;du[y]++;
    }
    if (n==1) {
        printf("1\n1 1\n");
        return 0;
    }
    if (n==2) {
        printf("1\n1 2\n");
        return 0;
    }
    for (int i = 1;i<= n;i++)
        if (du[i]!=1)
        {
            dfs(i,0);
            if (sons[i].size()==2)
            {
                anscnt++;
                anss[anscnt][0] = sons[i].front();
                sons[i].pop();
                anss[anscnt][1] = sons[i].front();
            }
            else {
                anscnt++;
                anss[anscnt][0] = sons[i].front();
                anss[anscnt][1] = i;
            }
            break;
        }
    printf("%d\n",anscnt);
    for (int i = 1;i<= anscnt;i++)
        printf("%d %d\n",anss[i][0],anss[i][1]);
    return 0;
}
```

D - Duration

签到题，把两个时间都换算成到 $0:0:0$ 的秒数，相减取绝对值。

```
#include<bits/stdc++.h>
#define ALL(x) (x).begin(),(x).end()
#define ll long long
```

```
#define ull unsigned long long
#define pii_ pair<int,int>
#define mp_ make_pair
#define pb push_back
#define fi first
#define se second
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define per(i,a,b) for(int i=(a);i>=(b);i--)
#define show1(a) cout<<#a<<" = "<<a<<endl
#define show2(a,b) cout<<#a<<" = "<<a<<"; "<<#b<<" = "<<b<<endl
using namespace std;
const ll INF = 1LL<<60;
const int inf = 1<<30;
const int maxn = 2e5+5;
inline void fastio() {ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);}



int main()
{
    //fastio();
    int a,b,c,x,y,z;
    scanf("%d:%d:%d",&a,&b,&c);
    scanf("%d:%d:%d",&x,&y,&z);
    ll t = a*3600 + b*60 + c;
    ll p = x*3600 + y*60 +z;
    ll ans = abs(t-p);
    printf("%lld\n",ans);
    return 0;
}
```

E - Exclusive OR

如果可以猜到结论/想到结论/打表观察可以发现对于$20\leq i$的$ans_i=ans_{i-2}$

那么就可以对于$i\leq 20$的时候fwt计算结果，其余的递推即可。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1<<18;
const int M = 2e5+5;
typedef long long ll;
int n,nans[N],one[N],ans[M];

void FWT(int *src) {
    for (int sz = 2;sz <= N;sz<<=1) {
        int step = sz >> 1;
```

```
        for (int i = 0;i< N;i+=sz) {
            for (int j = i;j < i+step;j++) {
                int a = src[j],b = src[j+step];
                src[j] = a+b;
                src[j+step] = a-b;
            }
        }
    }
}

void IFWT(int *src) {
    for (int sz = 2;sz <= N;sz<<=1) {
        int step = sz >> 1;
        for (int i = 0;i < N;i+=sz) {
            for (int j = i;j<i+step;j++) {
                int a = src[j],b = src[j+step];
                src[j] = (a+b) >> 1;
                src[j+step] = (a-b)>>1;
            }
        }
    }
}
int main()
{
    scanf("%d",&n);
    int x;
    for (int i = 1;i<= n;i++) {
        scanf("%d",&x);
        one[x]=1;
        nans[x]=1;
        if (x > ans[1]) ans[1] = x;
    }
    FWT(one);
    for (int i = 2;i<= 20;i++) {
        FWT(nans);
        for (int j = 0;j < N;j++)
            nans[j] = nans[j]*one[j];
        IFWT(nans);
        for (int j = 0;j < N;j++)
            if (nans[j])
                nans[j] = 1;
        for (int j = N-1;j>=0;j--)
            if (nans[j]) {
                ans[i] = j;
                break;
            }
    }
    for (int i = 21;i<= n;i++)
        ans[i] = ans[i-2];
    for (int i = 1;i<= n;i++)
```

```
        printf("%d ",ans[i]);
}
```

F - Fake Maxpooling

$n\times m$矩阵$a[i][j]=\gcd(i,j)$求所有$k\times k$子矩阵最大值的和

先对某一维单调队列（对于每行k列最大值缩为一格），之后另一维也单调队列□□$O(nm)$□gcd这个暴力要多个$\log n$□有点慢，但卡过去了，其实是应该记忆化

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int N=5e3+10;
int a[N][N],b[N][N];
int head,tail,q[N*10];
inline int gcd(int a,int b){return b?gcd(b,a%b):a;}
inline int lcm(int a,int b){return a*b/gcd(a,b);}
int main()
{
    int n,m,k;
    scanf("%d%d%d",&n,&m,&k);
    ll res=0;
    for(int i=1;i<=n;i++)
    {
        head=1,tail=0;
        for(int j=1;j<=m;j++)
        {
            a[i][j]=lcm(i,j);
            while(tail>=head&&j-q[head]>=k)head++;
            while(tail>=head&&a[i][j]>a[i][q[tail]])tail--;
            q[++tail]=j;
            if(j>=k)b[i][j-k+1]=a[i][q[head]];
        }
    }
    for(int j=1;j<=m-k+1;j++)
    {
        head=1,tail=0;
        for(int i=1;i<=n;i++)
        {
            while(tail>=head&&i-q[head]>=k)head++;
            while(tail>=head&&b[i][j]>b[q[tail]][j])tail--;
            q[++tail]=i;
            if(i>=k)res+=b[q[head]][j];
        }
```

```
    }
    printf("%lld\n",res);
    return 0;
}
```

G - Greater and Greater

给两个序列 A 和 B，求 A 的所有长度为 $|B|$ 的子串中，每一位都对应大于 B 的每一位的个数。

用 bitset 压位，先预处理出 S ，其中 $S[i][j]=1$ 表示 $A[i]>=B[j]$，并维护 A 的长度为 $|B|$ 的子串 a 的前缀与 B 的后缀的大小关系，考虑从后往前 dp，具体来说就是，$cur[i]=1$ 表示 a 的 $m-i$ 前缀每一位都对应大于等于 B 的 $m-i$ 后缀，从后往前滚动，每次向左移一位，把 $m-1$ 置为 1，再与 $S[i]$ 按位与，统计 $cur[0]=1$ 的个数就是答案。

```
#include<bits/stdc++.h>
#define ALL(x) (x).begin(),(x).end()
#define ll long long
#define ull unsigned long long
#define pii_ pair<int,int>
#define mp_ make_pair
#define pb push_back
#define fi first
#define se second
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define per(i,a,b) for(int i=(a);i>=(b);i--)
#define show1(a) cout<<#a<<" = "<<a<<endl
#define show2(a,b) cout<<#a<<" = "<<a<<"; "<<#b<<" = "<<b<<endl
using namespace std;
const ll INF = 1LL<<60;
const int inf = 1<<30;
const int maxn = 2e5+5;
inline void fastio() {ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);}

int a[150005],b[40005],c[40005],id[40005];
int n,m;
bitset<40000> S[40001],cur;

int main()
{
    fastio();
    cin>>n>>m;
    rep(i,0,n-1) cin>>a[i];
    rep(i,0,m-1) {cin>>b[i];c[i] = b[i];id[i]=i;}
    sort(c,c+m);
    sort(id,id+m,[](int x,int y){return b[x]<b[y];});
    rep(i,0,m-1){
```

```
        if(i>0) S[i] |= S[i-1];
        S[i][id[i]] = 1;
    }
    //rep(i,0,m-1) {rep(j,0,m-1) cout<<S[i][j]<<" ";cout<<endl;}
    int ans = 0;
    rep(i,0,m-1){
        int flag = 1;
        rep(j,0,m-i-1){
            if(a[n-m+j]<b[j+i]) {flag=0;break;}
        }
        cur[i] = flag;
    }
    ans += cur[0];
    per(i,n-m-1,0){ //show1(a[i]);
        int pos = upper_bound(c,c+m,a[i]) - c - 1; //show1(pos);
        cur >>= 1;
        cur[m-1] = 1;
        if(pos==-1) cur &= S[m];
        else cur &= S[pos];
        ans += cur[0];
    }
    cout<<ans<<endl;
    return 0;
}
```

H - Happy Triangle

题意：维护一个multiset□三种操作：添加删除查询。查询是查询对于x□multiset中是否存在两条边可以和其组成三角形。

分两种情况讨论，第一种情况是询问的边是最大的边，第二种情况是非最大边。对于最大边的情况可以发现找小于x的最大的两条边即可，这一部分直接用multiset就能维护。对于非最大边时，假设能组成三角形的另外两条边是a和b□那么一定有$x < b$且$b-a < x$□可以发现对于一个b□肯定是选择前驱作为a时$b-a$最小，所以我们需要做的就是写一个数据结构，使得可以维护当前点减去前驱点的后缀最小值。
仝multiset+离散化+线段树可以做到这一操作。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5+5;
const int INF = 2e9+5;
int tr[N<<2],id[N];
struct Node {
    int op,val;
}qur[N];
void Build(int p,int l,int r) {
    if (l==r) {
        tr[p] = INF;
```

```
        return ;
    }
    int mid = (l+r)>>1;
    Build(p<<1,l,mid);
    Build(p<<1|1,mid+1,r);
    tr[p] = min(tr[p<<1],tr[p<<1|1]);
}
void Update(int p,int l,int r,int a,int b) {
    if (l==r) {
        tr[p] = b;
        return ;
    }
    int mid = (l+r)>>1;
    if (a <= mid) Update(p<<1,l,mid,a,b);
    else Update(p<<1|1,mid+1,r,a,b);
    tr[p] = min(tr[p<<1],tr[p<<1|1]);
}
int Getans(int p,int l,int r,int a,int b) {
    if (l>=a&&r<=b) {
        return tr[p];
    }
    int mid = (l+r)>>1;
    int ans = INF;
    if (a<=mid) ans = min(ans,Getans(p<<1,l,mid,a,b));
    if (b>mid) ans = min(ans,Getans(p<<1|1,mid+1,r,a,b));
    return ans;
}
multiset<int> MS;
int c[N];
int main()
{
    MS.insert(-1);MS.insert(-1);MS.insert(INF);
    int q;
    scanf("%d",&q);
    for (int i = 1;i<= q;i++) {
        scanf("%d%d",&qur[i].op,&qur[i].val);
        id[i] = qur[i].val;
    }
    sort(id+1,id+q+1);
    int cnt = unique(id+1,id+q+1)-id-1;
    Build(1,1,cnt);
    for (int i = 1;i<= q;i++) {
        int x = qur[i].val;
        int pos = lower_bound(id+1,id+cnt+1,x)-id;
        if (qur[i].op == 1) {
            MS.insert(x);
            c[pos]++;
            if (c[pos] == 2) {
                Update(1,1,cnt,pos,0);
            } else if (c[pos] == 1) {
```

```
                Update(1,1,cnt,pos,(*MS.upper_bound(x))-x);
                auto it = MS.lower_bound(x);
                it--;
                int pre = *it;
                int prepos = lower_bound(id+1,id+cnt+1,pre)-id;
                if (c[prepos] == 1)Update(1,1,cnt,prepos,x-pre);
            }
        } else if(qur[i].op == 2) {
            MS.erase(MS.find(x));
            c[pos]--;
            if (c[pos] == 1) Update(1,1,cnt,pos,(*MS.upper_bound(x))-x);
            else if (c[pos] == 0) {
                Update(1,1,cnt,pos,INF);
                auto it = MS.lower_bound(x);
                it--;
                int pre = *it;
                int prepos = lower_bound(id+1,id+cnt+1,pre)-id;
                if (c[prepos] ==
1)Update(1,1,cnt,prepos,(*MS.lower_bound(x))-pre);
            }
        } else {
            bool flag = false;
            auto it = MS.lower_bound(x);
            int t1 = *it;
            int t2 = *(--it);
            int t3 = *(--it);
            if (t2+t3>x || t2+x>t1)flag = true;
            if (Getans(1,1,cnt,pos,cnt) < x) flag = true;
            if (flag)printf("Yes\n");
            else printf("No\n");
        }
    }
    return 0;
}
```

I - Interval

开局你有一个区间$[1,n]$□区间可以收缩和扩张□$[l,r]$和$[l+1,r]$□$[l,r]$和$[l,r-1]$间可以来回转换，但是到$l=r$时就不能再变了，所以不希望看到这种局面。给出m个限制□$l~r~dir~c$当$dir=L$时表示花费c可以限制$[l,r],[l+1,r]$之间的转换□$dir=R$时表示花费c可以限制$[l,r],[l,r-1]$之间的转换。问能够限制区间不可能变为$l=r$的最小花费。

我们把区间看作坐标，显然这是一个从$(1,n)$到$y=x$的最小割。和狼抓兔子□BZOJ1001□比较像，网格图转对偶图求最短路即最小割□$O(n^2\log n)$

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
#define pb push_back
using namespace std;
#define ll long long
#define INF 0x3f3f3f3f3f3f3f3f
#define pii pair<int,int>
#define idx(x,y) ((x-1)*(n-1)+y)
#define mp make_pair
const int N=505;
int n,m,s,t,head[N*N],tot=0,done[N*N];
ll dis[N*N];
struct Node
{
    int nxt,to;ll w;
    Node(int nxt=0,int to=0,ll w=0):nxt(nxt),to(to),w(w){}
}edges[N*N*10];
struct Node1
{
    int u;ll d;
    Node1(int u=0,ll d=0):u(u),d(d){}
    bool operator < (const Node1 &a) const {return d>a.d;}
};
void addedge(int u,int v,ll w)
{
    edges[++tot]=Node(head[u],v,w);
    head[u]=tot;
}
ll dij()
{
    priority_queue<Node1>q;
    memset(dis,0x3f,sizeof(dis));
    dis[s]=0,q.push(Node1(s,dis[s]));
    while(!q.empty())
    {
        int u=q.top().u;q.pop();
        if(done[u])continue;
        done[u]=1;
        for(int i=head[u];~i;i=edges[i].nxt)
        {
            int v=edges[i].to;
            if(dis[v]>dis[u]+edges[i].w)
            {
                dis[v]=dis[u]+edges[i].w;
                q.push(Node1(v,dis[v]));
            }
        }
    }
    return dis[t]==INF?-1:dis[t];
```

```
}
int main()
{
    memset(head,-1,sizeof(head));
    scanf("%d%d",&n,&m);
    s=0,t=(n-1)*(n-1)+1;
    for(int i=1;i<=m;i++)
    {
        int l,r,c;char dir[2];
        scanf("%d%d%s%d",&l,&r,dir,&c);
        int id1,id2;
        if(dir[0]=='L')
        {
            id1=idx(l,r),id2=idx(l,r-1);
            if(r==n)id1=s;
        }
        else
        {
            id2=idx(l-1,r-1),id1=idx(l,r-1);
            if(l==1)id2=t;
        }
        addedge(id1,id2,c),addedge(id2,id1,c);
    }
    printf("%lld\n",dij());
    return 0;
}
```

K - Keyboard Free

三个半径分别为r_1,r_2,r_3的同心圆上分别取一点构成三角形，求面积期望，保留一位小数。（懂了，保留一位小数就是增量乱搞qaq

暴力积分有点难，涉及到负面积之类的，所以掺杂一点乱搞成分比较好。第一维位置固定，第二维增量乱搞，第三维easy积分。

第一维取点G第二维在$[0,\pi]$间均匀枚举一定量的角度得E第三位积分算圆周到EG直线的距离即可得三角形面积。

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
using namespace std;
const double pi=acos(-1.0);
double r1,r2,r3;
double sqr(double x){return x*x;}
double f(double j)
```

```
{
    double EG=sqrt(sqr(r1)+sqr(r2)-2*cos(j)*r1*r2);
    double AGE=acos((sqr(r1)+sqr(EG)-sqr(r2))/2/r1/EG);
    double alp=asin(r1*sin(AGE)/r3);
    return r3*(alp*sin(alp)+cos(alp))*EG/pi;
}
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%lf%lf%lf",&r1,&r2,&r3);
        if(r1>r2)swap(r1,r2);
        if(r2>r3)swap(r2,r3);
        if(r1>r2)swap(r1,r2);
        double res=0,j=pi/10000.0;
        for(int i=0;i<5000;i++,j+=pi/5000.0)res+=f(j);
        printf("%.1lf\n",res/5000);
    }
    return 0;
}
```

比赛总结与反思

补完了！感觉这场题蛮正常的（就是隐约觉得都可做，然而没做对）。这种场的要义是不是不能陷入僵局quq卡题乃正常场之大忌），好几道没怎么看的题后来发现可能更值得入手？啊下周加油——Zars19

C题一开始想错了耽误了一些时间，想到了对的实现还很艰难呜呜呜呜该多写难题代码了FWT这个有点可惜，感觉要多做一些FWT题了 ——Infinity37