# 大数因子快速随机

```cpp
#include<bits/stdc++.h>
//Pollard_rho 大数因子分解快速随机算法
//Miller-Rabin 素数性测试算法
using namespace std;
ll qcpow_p(ll a,ll b,ll p){
        ll ret=1;
        for(;b;b>>=1,a=(__int128)a*a%p)
                if(b&1)ret=(__int128)ret*a%p;
        return ret;
}
bool Miller_Rabin(ll p){
        if(p<2)return 0;
        if(p==2||p==3)return 1;
        ll d=p-1,r=0;
        while(!(d&1))++r,d>>=1;
        for(ll k=0;k<10;++k){
                ll a=rand()%(p-2)+2;
                ll x=qcpow_p(a,d,p);
                if(x==1||x==p-1)continue;
                for(int i=0;i<r-1;++i){
                        x=(__int128)x*x%p;
                        if(x==p-1)break;
                }
                if(x!=p-1)return 0;
        }
        return 1;
}
ll Pollard_Rho(ll x){
        ll s=0,t=0;
        ll c=(ll)rand()%(x-1)+1;
        int step=0,goal=1;
        ll val=1;
        for(goal=1;;goal<<=1,s=t,val=1){
                for(step=1;step<=goal;step++){
                        t=((__int128)t*t+c)%x;
                        val=(__int128)val*abs(t-s)%x;
                        if(step%127==0){
                                ll d=__gcd(val,x);
                                if(d>1)return d;
                        }
                }
                ll d=__gcd(val,x);
                if(d>1)return d;
        }
}
```

```
int main(){
        ll t,n;
        cin >> t;
        while(t--){
                cin >> n;
                cout << Pollard_Rho(n) << endl;
        }
        return 0;
}
```

# 二部图最大匹配

```cpp
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
struct augment_path {
        vector<vector<ll> > g;
        vector<ll> pa;  // 匹配
        vector<ll> pb;
        vector<ll> vis;  // 访问
        ll n, m;         // 两个点集中的顶点数量
        ll dfn;          // 时间戳记
        ll res;          // 匹配数
        augment_path(ll _n, ll _m) : n(_n), m(_m) {
                assert(0 <= n && 0 <= m);
                pa = vector<ll>(n, -1);
                pb = vector<ll>(m, -1);
                vis = vector<ll>(n);
                g.resize(n);
                res = 0;
                dfn = 0;
        }
        void add(ll from, ll to) {
                assert(0 <= from && from < n && 0 <= to && to < m);
                g[from].push_back(to);
        }
        bool dfs(ll v) {
                vis[v] = dfn;
                for (ll u : g[v]) {
                        if (pb[u] == -1) {
                                pb[u] = v;
                                pa[v] = u;
                                return true;
                        }
                }
                for (ll u : g[v]) {
                        if (vis[pb[u]] != dfn && dfs(pb[u])) {
                                pa[v] = u;
                                pb[u] = v;
                                return true;
                        }
                }
                return false;
        }
        ll solve() {
                while (true) {
                        dfn++;
                        ll cnt = 0;
```

```
                        for (ll i = 0; i < n; i++) {
                                if (pa[i] == -1 && dfs(i)) {
                                        cnt++;
                                }
                        }
                        if (cnt == 0) {
                                break;
                        }
                        res += cnt;
                }
                return res;
        }
};
int main(){
    ll n,m;
    cin >> n >> m;
    augment_path G(n,n);
    ll u,v;
    for(ll i=0;i<m;i++){
        cin >> u >> v;
        G.add(u,v);
    }cout << G.solve() << endl;
    return 0;
}
```

# 二维计算几何

```cpp
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<ll,ll> pll;

namespace DEFINITION
{
    #define scanfll(a) scanf("%lld",&a)
    #define lowbit(x) ((x)&(-(x)))
    #define RESET(A) memset(A,0,sizeof(A))
    #define ALL(A) A.begin(),A.end()
    #define SORT(A) sort(ALL(A))
    #define Presentation(i,r) " \n"[i==r]
    #define FORLL(i,l,r) for(ll i=l;i<=r;i++)
    #define FORLL_rev(i,r,l) for(ll i=r;i>=l;i--)
    #define Get_Mod(a) (((a)+MOD)%MOD)
    #define NO "NO\n"
    #define YES "YES\n"
}using namespace DEFINITION;

/*----------Consts----------*/
const double eps = 1e-8;
const double inf = 1e20;
const double pi = acos(-1.0);
/*----------Consts----------*/
int sgn(double x) {if(fabs(x)<eps) return 0;if(x<0) return -1;return 1;}
inline double sqr(double x) {return x*x;}

struct Point{
    double x,y;
    Point(){}//Empty Point
    Point(double _x,double _y){x = _x;y = _y;}//Point

    void input(){cin >> x >> y;}
    #ifndef print_float
        #define print_float(value,digit)
        cout << fixed << setprecision(digit) << value;
    #endif
    void output(int digit){
        print_float(x,digit);
        cout << ',';
        print_float(y,digit);
        cout << '\n';
    }void output(){output(7);}

    bool operator == (Point b)const {return sgn(x-b.x) == 0 && sgn(y-b.y) == 0;}
```

```
    bool operator <  (Point b)const {return sgn(x-b.x)== 0?sgn(y-b.y)<0:x<b.x;}
    bool operator >  (Point b)const {return sgn(x-b.x)== 0?sgn(y-b.y)>0:x>b.x;}

    Point operator - (const Point &b)const{return Point(x-b.x,y-b.y);}
    //相减(向量): A-B=BA
    Point operator + (const Point &b)const{return Point(x+b.x,y+b.y);}//向量和

    double operator * (const Point &b)const{return x*b.x + y*b.y;}//点积
    double operator ^ (const Point &b)const{return x*b.y - y*b.x;}//叉积

    double len(){return hypot(x,y);}//向量长度
    double len2(){return x*x + y*y;}//向量长度平方

    double distance(Point p){return hypot(x-p.x,y-p.y);}//与另一点的距离

    Point operator *(const double &k)const{return Point(x*k,y*k);}
    Point operator /(const double &k)const{return Point(x/k,y/k);}

    //计算 pa 和 pb 的夹角, 就是求这个点看 a,b 所成的夹角
    double rad(Point a,Point b) {
        Point p = *this;
        return fabs(atan2(fabs((a-p)^(b-p)),(a-p)*(b-p)));}

    Point trunto(double r){
        double l = len();if(!sgn(l))return *this;
        r /= l;return Point(x*r,y*r);}//化为长度为 r 的向量

    Point rotleft(){return Point(-y,x);}//逆时针旋转 90 度
    Point rotright(){return Point(y,-x);}//顺时针旋转 90 度

    //绕着 p 点逆时针旋转angle(弧度制)
    Point rotate(Point p,double angle){
        Point v = (*this) - p;double c = cos(angle), s = sin(angle);
        return Point(p.x + v.x*c - v.y*s,p.y + v.x*s + v.y*c);}

};

//计算凸包
vector<Point> Convex_Hull(vector<Point> pvec){
    vector<Point> ch;
    ll n=pvec.size();SORT(pvec);
    vector<ll> stk(n+1);ll top=0;stk[++top]=0;
    vector<bool> used(n+1,false);
    FORLL(i,1,n-1){
        while(top>1
        &&(pvec[stk[top]]-pvec[stk[top-1]]).operator^(pvec[i]-pvec[stk[top]])<=0)
            used[stk[top--]]=false;
        stk[++top]=i;
        used[i]=true;
    }
    ll tmp = top;

```

```
    FORLL_rev(i,n-2,0) if (!used[i]){
        while(top>tmp
        &&(pvec[stk[top]]-pvec[stk[top-1]]).operator^(pvec[i]-pvec[stk[top]])<=0)
           used[stk[top--]]=false;
        stk[++top]=i;
        used[i]=true;
    }
    FORLL(i,1,top) ch.emplace_back(pvec[stk[i]]);
    return ch;
}
```

# 线性筛+欧拉函数

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

const ll MAXN = 1000;
bool check[MAXN+10];
ll phi[MAXN+10];
ll prime[MAXN+10];
ll tot;//素数的个数
void Phi_and_Prime_Table(ll N){
    memset(check,false,sizeof(check)); phi[1] = 1; tot = 0;
    for(ll i = 2; i <= N; i++) {if( !check[i] ) {prime[tot++] = i; phi[i] = i-1; }
        for(ll j = 0; j < tot; j++){
            if(i * prime[j] > N) break;
            check[i * prime[j]] = true;
            if( i % prime[j] == 0){
                phi[i * prime[j]] = phi[i] * prime[j]; break; }
            else {phi[i * prime[j]] = phi[i] * (prime[j] - 1);}}}}
```

# int128

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;


struct int128{
    __int128_t value;

    int128():value(0){}
    int128(ll _val):value(_val){}
    int128(__int128_t _val):value(_val){}

    static int128 read() {
        string input;cin >> input;
        bool isNegative = false;
        if (input[0] == '-') {isNegative = true;input = input.substr(1);}
        __int128_t result=0;
        for (char c : input) {result = result * 10 + (c - '0');}
        if (isNegative) {result = -result;}
        return int128(result);
    }

    void print() const {
        __int128_t x = value;
        if (x < 0) {putchar('-');x = -x;}
        if (x > 9) {int128(x / 10).print();}
        putchar(x % 10 + '0');
    }

    int128 operator + (const int128 &b)const{return value+b.value;}
    int128 operator - (const int128 &b)const{return value-b.value;}
    int128 operator * (const int128 &b)const{return value*b.value;}
    int128 operator / (const int128 &b)const{return value/b.value;}
};
int main(){
    int128 i;
    i=0x3fffffffffffffff;
    ll t;cin >> t;
    ll x;
    while(t--){
        cin >> x;i=i/x;
        i.print();putchar('\n');
    }
    return 0;
}
```

# 并查集

```cpp
struct DSU{
    vector<ll> parents,size;

    explicit DSU(ll n):parents(n+1),size(n+1,1) {iota(ALL(parents),0);}

    ll find(ll x){ return (parents[x]==x)?x:(parents[x]=find(parents[x])); }

    void merge(ll a,ll b){
        a=find(a);b=find(b);
        if(a==b) return ;
        if (size[a]>size[b]) swap(a,b);
        parents[a]=b;
        size[a]+=size[b];
    }
};
```

# 树状数组

```
#define lowbit(x) ((x)&(-(x)))//取最后一个1所在位置的权值
struct BITree{//树状数组，下标i从1开始
    vector<ll> Data;

    explicit BITree(ll n):Data(n*2+5,0) {}

    void update(ll i,ll dif)
    {//给予i增量dif,维护树状数组，O(logn)
        while(i<Data.size()){
            Data[i]+=dif;
            i+=lowbit(i);
        }
    }

    ll presum(ll i)
    {//查询前缀和sum[i]，O(logn)
        ll sum=0;
        while(i){
            sum+=Data[i];
            i-=lowbit(i);
        }
        return sum;
    }

    ll query(ll l,ll r){//查询区间和
        return presum(r)-presum(l-1);
    }

    ll operator[](ll index){//下标调用前缀和
        //if(index>(Data.size()-5/2)||index<0)
        //    throw out_of_range("Index out of range.");
        return presum(index);
    }
};
int main()
{
    ll n,t,tt=0;
    cin >> n;
    BITree bt(n);
    for(ll i=1;i<=n;i++){
        cin >> t;
        bt.update(i,t);//维护原数组，实现单点修改，区间查询
        /*
        bt.update(i,t-tt);tt=t;
        维护差分数组，实现区间修改，单点查询
        对区间[l,r]的修改变为update(l,dif);update(r+1,-dif);
        对元素a[i]的查询变为presum(i);
```

```
        */
    }//建树O(nlogn)
    ll l,r;
    cin >> l >> r;
    cout << bt.query(l,r) << endl;
    ll i,x;
    cin >> i >> x;
    bt.update(i,x);
    cin >> l >> r;
    cout << bt.query(l,r) << endl;
    return 0;
}
```

# 归并排序与逆序对数计算

```
long long mergeAndCount(vector<long long>& arr,
long long l, long long m, long long r) {
    vector<long long> temp(r - l + 1);
    long long invCount = 0;
    long long i = l, j = m + 1, k = 0;
    while (i <= m && j <= r) {
        if (arr[i] <= arr[j]) temp[k++] = arr[i++];
        else{
            temp[k++] = arr[j++];
            invCount += m - i + 1;}
    }
    while (i <= m) temp[k++] = arr[i++];
    while (j <= r) temp[k++] = arr[j++];
    for (long long p = 0; p < temp.size(); p++) arr[l + p] = temp[p];

    return invCount;
}

long long mergeSortAndCount(vector<long long>& arr, long long l, long long r) {
    long long invCount = 0;
    if (l < r) {
        long long m = l + (r - l) / 2;
        invCount += mergeSortAndCount(arr, l, m);
        invCount += mergeSortAndCount(arr, m + 1, r);
        invCount += mergeAndCount(arr, l, m, r);
    }
    return invCount;
}
```

# 参考程序

## 1.树形DP

```
#define N 2005
vector<vector<ll>> T;
vector<ll> cost;
vector<ll> DFS_vec;
void DFS(ll node){
    DFS_vec.emplace_back(node);
    for(auto i:T[node]) DFS(i);
}//获取DFS遍历序列
int solve()
{
    ll n;
    cin >> n;
    T.clear();T.resize(n+1);
    cost.clear();cost.resize(n+1);

    FORLL(i,1,n) cin >> cost[i];
    ll u,v;
    FORLL(i,1,n-1){
        cin >> u >> v;
        T[u].emplace_back(v);
    }

    DFS_vec.clear();
    DFS(1);
    reverse(ALL(DFS_vec));//逆序
    //print_vec(DFS_vec);

    ll dp[N][3]={0};
    for(auto i:DFS_vec)
    {
        dp[i][0]=cost[i];
        dp[i][1]=0;
        dp[i][2]=0;
        for(auto j:T[i])
        {
            dp[i][0]+=min({dp[j][0],dp[j][1],dp[j][2]});
            dp[i][2]+=dp[j][1];
        }

        if(T[i].empty()) dp[i][1]=INF;
        else{
            int flag=1;ll inc=INF;
            for(auto j:T[i])
```

```
            {
                if(dp[j][0]<=dp[j][1]) {dp[i][1]+=dp[j][0];flag=0;}
                else dp[i][1]+=dp[j][1];
            }
            if(flag){
                for(auto j:T[i]) inc=min({inc,dp[u][0]-dp[u][1]});
                dp[i][1]+=inc;
            }
        }
    }

    ll re=min({dp[1][0],dp[1][1]});
    cout << re << endl;

    return 0;
}
```

# 2.度与公共邻居问题状态压缩

```
ll C[1005][1005]={0};
//在主函数中预处理组合数C，代码略
int solve()
{
    ll n,m;
    cin >> n >> m;
    bitset<1005> G[1005];
    int deg[1005]={0};
    ll u,v;
    FORLL(i,1,m){
        cin >> u >> v;
        G[u].set(v);
        G[v].set(u);
        deg[u]++;deg[v]++;
    }
    ll re=0,nbr,deg1,deg2;
    FORLL(i,1,n) if(deg[i]>=4){
        FORLL(j,i+1,n) if(j-i&&deg[j]>=4){
            deg1=deg[i]-G[i][j];
            deg2=deg[j]-G[i][j];
            //如果vi,vj直接相连，这条边是不能构入的
            nbr=(G[i]&G[j]).count();
            if(nbr>=4){
                if(deg1>=6) re=add(re,mul(C[nbr][4],C[deg1-4][2]));
                if(deg2>=6) re=add(re,mul(C[nbr][4],C[deg2-4][2]));
            }
        }
    }
    cout << re << endl;
    return 0;
}
```

# 3.Floyd与有向图最短回路计数

```
const long long INF=1e18;
#define N 505
ll dist[N][N]={0},cnt[N][N]={0},ori[N][N]={0};
ll mindist=INF,cntmin=0;
void RESET_G(ll n){
    FORLL(i,1,n)
        FORLL(j,1,n){
        ori[i][j]=0;
        dist[i][j]=(i==j?0:INF);
        cnt[i][j]=0;
        }
    mindist=INF,cntmin=0;
}//重置
void Floyd(ll n){
    FORLL(k,1,n)
        FORLL(i,1,n){
            FORLL(j,1,n){//Floyd
                if(dist[i][j]>dist[i][k]+dist[k][j]){
                    dist[i][j]=dist[i][k]+dist[k][j];
                    cnt[i][j]=mul(cnt[i][k],cnt[k][j]);
                    //cnt[i][j]=cnt[i][k]*cnt[k][j]
                    //方案数为两段方案数的乘积，更新同步
                }else if(dist[i][j]==dist[i][k]+dist[k][j]){
                    addto(cnt[i][j],mul(cnt[i][k],cnt[k][j]));
                    //cnt[i][j]+=cnt[i][k]*cnt[k][j]
                    //相等则方案数相加
                }
            }
            if(i<k&&ori[k][i]){//假设到k为以i为起点的单向环上最大点
                if(ori[k][i]+dist[i][k]<mindist)
                    {mindist=ori[k][i]+dist[i][k];cntmin=cnt[i][k];}
                else if(ori[k][i]+dist[i][k]==mindist)
                    addto(cntmin,cnt[i][k]);//cntmin+=cnt[i][k]
        }
    }
}
void solve()
{
    ll n,m;
    cin >> n >> m;
    RESET_G(n);
    ll u,v,w;
    FORLL(i,1,m){
        cin >> u >> v >> w;
        ori[u][v]=w;
        dist[u][v]=w;
        cnt[u][v]=1;

```

```
    }
    Floyd(n);
    if(cntmin) cout << mindist << ' ' << cntmin << endl;
    else cout << "-1 -1" << endl;
}
```

# 记忆化搜索

```
unordered_map<ll,ll> mp;
ll dfs(ll n){
    if(n<=1) return 1-n;
    if(mp[n]) return mp[n];
    ll t1,t2;
    t1=n%2+1+dfs(n/2);
    t2=n%3+1+dfs(n/3);
    return mp[n]=min(t1,t2);
}
void solve()
{
    ll n;cin >> n;
    cout << dfs(n) << endl;
}
```