

Contents

1	Math	3
1.1	Quick power	3
1.2	Matrix multiply	3
1.3	Matrix quick power	4
1.4	Primes	4
1.5	Moebius function	4
1.6	Linear basis	6
1.7	Fast Fourier Transform	7
1.8	Simplex	7
2	Graph theory	8
2.1	Kruskal	8
2.2	Dijkstra	9
2.3	Bellman-Ford	10
2.4	Topological sorting	11
2.5	Edmonds-Karp	12
2.6	Dinic	13
2.7	Min cost max flow (repeated Dijkstra)	14
2.8	Hungary	16
2.9	Kuhn-Munkres	16
2.10	SCC decomposition	18
2.11	2-SAT	19
2.12	LCA (doubling)	20
2.13	LCA (RMQ)	21
2.14	LCA (Tarjan)	22
2.15	Diameter of tree	23
2.16	Maximum density subgraph	24
3	Data structures	26
3.1	Discretization	26
3.2	Segment tree (add)	27
3.3	Segment tree (add, set)	27
3.4	Segment tree (add, multiply)	30
3.5	Segment tree (sum of sums)	31
3.6	Heap	32
3.7	RMQ	32
3.8	Weighted union find	33
3.9	Treap	33
3.10	Cartesian tree	36
3.11	Expression tree	37
3.12	Persistent segment tree	37
3.13	Chain decomposition	39

4	String	40
4.1	KMP	40
4.2	Aho-Corasick automaton	41
4.3	Suffix array	42
4.4	Suffix array (SA-IS)	46
4.5	Manacher	51
4.6	Palindromic automaton	51
5	Geometry	52
5.1	2-D vector	52
5.2	2-D line	53
5.3	Area of polygon	54
5.4	Point-polygon distance	54
5.5	Segment+circle intersection	55
5.6	Half-plane intersection	55
5.7	Convex hull	56
5.8	Rotating calipers	57
6	IO	58
6.1	Fast read	58
6.2	Fast input (fin)	58

1 Math

1.1 Quick power

```
int power(int x,int e)
{
    if(!e) return 1;
    LL res=power(x,e/2);
    return (e%2 ? res*res%M*x%M : res*res%M);
}
```

1.2 Matrix multiply

```
struct MAT
{
    int v[SZ][SZ];
    int sz;
    MAT()
    {
        memset(v,0,sizeof(v));
    }
    MAT(int s,int k)
    {
        sz=s;
        int i;
        if(k==0) memset(v,0,sizeof(v));
        else if(k==1)
        {
            memset(v,0,sizeof(v));
            for(i=0;i<sz;i++) v[i][i]=1;
        }
    }
    MAT operator*(const MAT &rhs) const
    {
        int i,j,k;
        MAT tmp(sz,0);
        for(i=0;i<sz;i++)
            for(j=0;j<sz;j++)
                for(k=0;k<sz;k++)
                    tmp.v[i][j]+=v[i][k]*rhs.v[k][j];
        return tmp;
    }
    MAT operator^(int e) const
    {
        int i;
        MAT tmp(sz,1);
        for(i=1;i<=e;i++)
            tmp=tmp*(*this);
        return tmp;
    }
};
```

```
/*To multiply a matrix by a vector, first make the vector into a square matrix!*/
```

1.3 Matrix quick power

```
MAT mpw(MAT m,int e,int sz)
{
    if(!e) return MAT(sz,1);
    MAT t=mpw(m,e/2,sz);
    if(e%2) return t*t*m;
    else return t*t;
}
```

1.4 Primes

```
#include <stdio.h>
#include <string.h>
#define N 50000000

typedef long long LL;
LL f[N+10];

int main(void)
{
    freopen("prime.txt","w",stdout);
    memset(f,0,sizeof(f));
    LL i,j,prod=1;
    for(i=2;i<=N;i++)
        if(!f[i])
        {
            if(i<=60)
            {
                prod*=i;
                printf("%lld\t\t\t\t\tprod:\t%lld\n",i,prod);
            }
            else printf("%lld\n",i);
            for(j=i;i*j<=N;j++) f[i*j]=1;
        }
    return 0;
}
```

1.5 Moebius function

```
void moebius(int n)
{
    vector<int> p;
    int i,j,t;
    for(t=n,i=2;i*i<=n;i++)
    {
```

```

        if(t%i==0) p.push_back(i);
        while(t%i==0) t/=i;
    }
    if(t>1) p.push_back(t);
    int m=p.size(),x,y;
    for(mu.clear(),i=0;i<1<<m;i++)
    {
        for(x=1,y=1,j=0;j<m;j++)
            if(i&(1<<j))
            {
                x*=p[j];
                y*=-1;
            }
        mu[x]=y;
    }
}

/* count of non-periodic strings:

#include <cstdio>
#include <cstring>
#include <algorithm>
#include <map>
#include <vector>
using namespace std;
typedef long long LL;
const int M=1000000007;
map<int,int> mu;
int T,n;
void moebius(int n);
int power(int x,int e);

int main()
{
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        moebius(n);
        int ans=0;
        map<int,int>::iterator iter;
        for(iter=mu.begin();iter!=mu.end();++iter)
            ans=((ans+power(10,n/iter->first)*iter->second)%M+M)%M;
        printf("%d\n",ans);
    }
    return 0;
}

int power(int x,int e)
{
    if(!e) return 1;
    LL res=power(x,e/2);
    return (e%2 ? res*res%M*x%M : res*res%M);
}

```

```

void moebius(int n)
{
    vector<int> p;
    int i,j,t;
    for(t=n,i=2;i*i<=n;i++)
    {
        if(t%i==0) p.push_back(i);
        while(t%i==0) t/=i;
    }
    if(t>1) p.push_back(t);
    int m=p.size(),x,y;
    for(mu.clear(),i=0;i<1<<m;i++)
    {
        for(x=1,y=1,j=0;j<m;j++)
            if(i&(1<<j))
            {
                x*=p[j];
                y*=-1;
            }
        mu[x]=y;
    }
}
*/

```

1.6 Linear basis

```

int l,b[BIT_LENGTH];
// l (maximal bit length) needs to be initialized; b[] should be
// set to 0 initially

void insert(int x)
{
    int i,j;
    for(i=l-1;i>=0;i--)
    {
        if(!(x&(1<<i))) continue;
        if(b[i]) x^=b[i];
        else
        {
            for(j=0;j<i;j++) x^=b[j];
            for(j=i+1;j<l;j++)
                if(b[j]&(1<<i)) b[j]^=x;
            b[i]=x;
            break;
        }
    }
}

int find(int x)
{
    int i;
    for(i=l-1;i>=0;i--)
        if(x&(1<<i)) x^=b[i];
    return x ? 0 : 1;
}

```

```
}
```

1.7 Fast Fourier Transform

```
void build(const CPLX x[],int &i,int s,int d,int l,CPLX x2[])
{
    if(l==1)
    {
        x2[s]=x[i++];
        return;
    }
    build(x,i,s,d*2,l/2,x2);    build(x,i,s+d,d*2,l/2,x2);
}

void fft(const CPLX x[],int n,CPLX y[],int o)    // o=1: FFT;  o
=-1: Inverse FFT
{
    int t=0;
    build(x,t,0,1,n,y);
    int i,j,k;
    CPLX t1,t2,w,w0;
    for(i=1;i<n;i*=2)
        for(j=0;j<n;j+=i*2)
            for(k=0,w=1,w0=polar(1.0,PI/i*o);k<i;k++,w=w*w0)
            {
                t1=y[j+k];    t2=w*y[j+k+i];
                y[j+k]=t1+t2;    y[j+k+i]=t1-t2;
            }
    if(o>0) return;
    for(i=0;i<n;i++) y[i]/=n;
}
```

1.8 Simplex

```
int id[N_ROW+N_COL];

void pivot(double a[][N_COL],int x,int y,int r,int c)
{
    int i,j;
    double t;
    std::swap(id[x+c],id[y]);
    t=-a[x][y];    a[x][y]=-1;
    for(i=0;i<=c;i++) a[x][i]/=t;
    for(i=0;i<=r;i++)
    {
        if(i==x) continue;
        t=a[i][y];    a[i][y]=0;
        for(j=0;j<=c;j++)
            a[i][j]+=t*a[x][j];
    }
}
```

```

double simplex(double a[][N_COL],double v[],int r,int c)
{
    int i,x,y;
    double t;
    for(i=1;i<=r+c;i++) id[i]=i;
    for(;;)
    {
        for(x=0,i=1;i<=r;i++)
            if(a[i][0]<=-EPS)
            {
                x=i;
                break;
            }
        if(!x) break;
        for(y=0,i=1;i<=c;i++)
            if(a[x][i]>EPS)
            {
                y=i;
                break;
            }
        if(!y) return NAN; // Unfeasible
        pivot(a,x,y,r,c);
    }
    for(;;)
    {
        for(y=0,i=1;i<=c;i++)
            if(a[0][i]>EPS)
            {
                y=i;
                break;
            }
        if(!y) break;
        for(x=0,t=INFINITY,i=1;i<=r;i++)
            if(a[i][y]<=-EPS && -a[i][0]/a[i][y]<t)
                x=i,t=-a[i][0]/a[i][y];
        if(!x) return NAN; // Unbounded
        pivot(a,x,y,r,c);
    }
    for(i=1;i<=c;i++)
        if(id[i]<=c) v[id[i]]=0;
    for(i=1;i<=r;i++)
        if(id[i+c]<=c) v[id[i+c]]=a[i][0];
    return a[0][0];
}

```

2 Graph theory

2.1 Kruskal

```

struct EDGE
{
    int s,t,w;
}

```



```

    bool operator<(const EDGE &rhs) const
    {
        return w<rhs.w;
    }
};
EDGE edge[N_EDGE];
int m,sz,first[N_NODE],nxt[N_EDGE*2],tail[N_EDGE*2],len[N_EDGE*2],f[N_NODE];

int find(int u)
{
    return (f[u]==u ? u : (f[u]=find(f[u])));
}

void addedge(int u,int v,int l)
{
    sz++;
    tail[sz]=v; len[sz]=l;
    nxt[sz]=first[u]; first[u]=sz;
}

int Kruskal()
{
    std::sort(edge+1,edge+m+1);
    int i,ans=0;
    for(i=1;i<=n;i++) f[i]=i;
    for(sz=0,i=1;i<=m;i++)
        if(find(edge[i].s)!=find(edge[i].t))
        {
            ans+=edge[i].w;
            f[find(edge[i].s)]=find(edge[i].t);
            addedge(edge[i].s,edge[i].t,edge[i].w);
            addedge(edge[i].t,edge[i].s,edge[i].w);
        }
    return ans;
}

```

2.2 Dijkstra

```

#include <functional>

using namespace std;
typedef long long LL;
typedef pair<LL,LL> P;
LL sz,first[N_NODE],nxt[N_EDGE],tail[N_EDGE],len[N_EDGE],dist[N_NODE];

void addedge(LL u,LL v,LL w)
{
    sz++;
    tail[sz]=v; len[sz]=w;
    nxt[sz]=first[u]; first[u]=sz;
}

```

```

void dijkstra(LL s)
{
    memset(dist,0x3f,sizeof(dist));
    priority_queue<P,vector<P>,greater<P>> q;
    q.push(P{0,s});
    LL u,v,e;
    while(!q.empty())
    {
        P t=q.top(); q.pop();
        if(t.first>=dist[t.second]) continue;
        u=t.second;
        dist[u]=t.first;
        for(e=first[u];e=e=nxt[e])
        {
            v=tail[e];
            if(dist[u]+len[e]<dist[v])
                q.push(P{dist[u]+len[e],v});
        }
    }
}

LL n,first[N_NODE],nxt[N_EDGE],tail[N_EDGE],len[N_EDGE],used[
N_NODE],dist[N_NODE]

void dijkstra2(LL s)    // without priority-queue
{
    memset(used,0,sizeof(used));
    memset(dist,0x3f,sizeof(dist));
    dist[s]=0;
    LL i,j,mind,mark,v,e;
    for(i=1;i<=n;i++)    // n is the number of nodes
    {
        for(mind=INF,j=1;j<=n;j++)
            if(!used[j] && dist[j]<mind)
            {
                mind=dist[j];
                mark=j;
            }
        used[mark]=1;
        for(e=first[mark];e=e=nxt[e])
        {
            v=tail[e];
            if(dist[mark]+len[e]<dist[v])
                dist[v]=dist[mark]+len[e];
        }
    }
}

```

2.3 Bellman-Ford

```

int nv,ne,dist[N_NODE];

void bellman_ford(int s)
{

```

```

int i,u,e;
for(i=1;i<=nv;i++) dist[i]=INF;
dist[s]=0;
for(i=1;i<=nv-1;i++)
    for(u=1;u<=nv;u++)
        for(e=first[u];e;e=nxt[e])
            dist[tail[e]]=std::min(dist[tail[e]],dist[u]+len
                                   [e]);
}

```

2.4 Topological sorting

```

int sz,first[N_NODE],nxt[N_EDGE],tail[N_EDGE],ideg[N_NODE],vis[
    N_NODE],topo[N_NODE],q[N_NODE];

void addedge(int u,int v)
{
    tail[++sz]=v;
    nxt[sz]=first[u]; first[u]=sz;
    ideg[v]++;
}

int toposort()
{
    int front=1,rear=0,i,u,v,e;
    memset(vis,0,sizeof(vis));
    for(i=1;i<=n;i++)
        if(!ideg[i])
        {
            q[++rear]=i;
            vis[i]=1;
        }
    i=0;
    while(front<=rear)
    {
        u=q[front++];
        topo[++i]=u;
        for(e=first[u];e;e=nxt[e])
        {
            v=tail[e];
            ideg[v]--;
            if(!ideg[v])
            {
                q[++rear]=v;
                vis[v]=1;
            }
        }
    }
    return i==n;
}

```

2.5 Edmonds-Karp

```
int nv, ne, first[N_NODE], nxt[N_EDGE*2], tail[N_EDGE*2], rev[N_EDGE*2], from[N_EDGE*2], que[N_NODE];
double cap[N_EDGE*2], f[N_NODE];

void addedge(int u, int v, double c)
{
    ne++;
    tail[ne]=v; cap[ne]=c;
    nxt[ne]=first[u]; first[u]=ne;
    ne++;
    tail[ne]=u; cap[ne]=0;
    nxt[ne]=first[v]; first[v]=ne;
    rev[ne]=ne-1; rev[ne-1]=ne;
}

double bfs(int s, int t)
{
    int front=1, rear=0, u, v, e;
    memset(f, 0, sizeof(f));
    que[++rear]=s;
    f[s]=INF;
    while(front<=rear)
    {
        u=que[front++];
        if(u==t) break;
        for(e=first[u]; e; e=nxt[e])
        {
            v=tail[e];
            if(f[v] || fabs(cap[e])<EPS) continue;
            que[++rear]=v;
            f[v]=std::min(f[u], cap[e]);
            from[v]=e;
        }
    }
    return f[t];
}

double flow(int s, int t)
{
    double res=0;
    int u;
    while(bfs(s, t))
    {
        for(u=t; u!=s; u=tail[rev[from[u]]])
        {
            cap[from[u]]-=f[t];
            cap[rev[from[u]]]+=f[t];
        }
        res+=f[t];
    }
    return res;
}
```

2.6 Dinic

```
// pay attention to the initial value of cur[]!
int ne,nv,first[N_NODE],nxt[N_EDGE*2],tail[N_EDGE*2],cap[N_EDGE*2],rev[N_EDGE*2],cur[N_NODE],level[N_NODE],que[N_NODE];

void addedge(int u,int v,int c)
{
    ne++;
    tail[ne]=v; cap[ne]=c;
    nxt[ne]=first[u]; first[u]=ne;
    ne++;
    tail[ne]=u; cap[ne]=0;
    nxt[ne]=first[v]; first[v]=ne;
    rev[ne]=ne-1; rev[ne-1]=ne;
}

void bfs(int s)
{
    memset(level,-1,sizeof(level));
    level[s]=0;
    int front=1,rear=1,u,v,e;
    que[1]=s;
    while(front<=rear)
    {
        u=que[front++];
        for(e=first[u];e;e=nxt[e])
        {
            v=tail[e];
            if(cap[e]==0 || level[v]>=0) continue;
            level[v]=level[u]+1;
            que[++rear]=v;
        }
    }
}

int dfs(int u,int t,int f)
{
    if(u==t) return f;
    int v,d,&e=cur[u];
    for(;e;e=nxt[e])
    {
        v=tail[e];
        if(cap[e]==0 || level[u]>=level[v]) continue;
        d=dfs(v,t,std::min(f,cap[e]));
        if(d>0)
        {
            cap[e]-=d;
            cap[rev[e]]+=d;
            return d;
        }
    }
    return 0;
}
```

```

int dinic(int s,int t)
{
    int ans=0,i,f;
    for(;;)
    {
        bfs(s);
        if(level[t]<0) break;
        for(i=1;i<=nv;i++)
            cur[i]=first[i]; // pay attention to the initial
                             value of cur[]!
        while((f=dfs(s,t,INF))>0) ans+=f;
    }
    return ans;
}

```

2.7 Min cost max flow (repeated Dijkstra)

```

using namespace std;
typedef pair<int,double> P;
int nv,ne,first[N_NODE],nxt[N_EDGE*2],tail[N_EDGE*2],cap[N_EDGE
*2],rev[N_EDGE*2],from[N_NODE],used[N_NODE];
double len[N_EDGE*2],dist[N_NODE],h[N_NODE];

void addedge(int u,int v,int c,double l)
{
    ne++;
    tail[ne]=v; cap[ne]=c; len[ne]=l;
    nxt[ne]=first[u]; first[u]=ne;
    ne++;
    tail[ne]=u; cap[ne]=0; len[ne]=-l;
    nxt[ne]=first[v]; first[v]=ne;
    rev[ne]=ne-1; rev[ne-1]=ne;
}

void bellman_ford(int s)
{
    int i,u,v,e;
    for(i=1;i<=nv;i++) dist[i]=INF;
    dist[s]=0;
    memset(from,0,sizeof(from));
    for(i=1;i<=nv-1;i++)
        for(u=1;u<=nv;u++)
            for(e=first[u];e;nxt[e])
            {
                if(!cap[e]) continue;
                v=tail[e];
                if(dist[u]+len[e]<dist[v])
                {
                    dist[v]=dist[u]+len[e];
                    from[v]=e;
                }
            }
}

```

```

void search(int s)
{
    memset(used,0,sizeof(used));
    int i,j,mark,v,e;
    double mind;
    for(i=1;i<=nv;i++) dist[i]=INF;
    dist[s]=0;
    memset(from,0,sizeof(from));
    for(i=1;i<=nv;i++)
    {
        for(mind=INF,j=1;j<=nv;j++)
            if(!used[j] && dist[j]<mind-EPS)
            {
                mind=dist[j];
                mark=j;
            }
        used[mark]=1;
        for(e=first[mark];e;e=nxt[e])
        {
            v=tail[e];
            if(cap[e] && dist[mark]+len[e]+h[mark]-h[v]<dist[v]-EPS)
            {
                dist[v]=dist[mark]+len[e]+h[mark]-h[v];
                from[v]=e;
            }
        }
    }
}

P mcmf(int s,int t)
{
    double res=0;
    int i,u,flow=0,f;
    memset(h,0,sizeof(h));
    for(;;)
    {
        search(s);

        // When there are negative-weight edges initially, call
        // bellman_ford(s) in the FIRST ITERATION.

        for(i=1;i<=nv;i++)
            h[i]+=dist[i];
        if(!from[t]) break;

        // For Min Cost Feasible Flow:
        // if(!from[t] || h[t]>=0) break;

        for(f=INF,u=t;u!=s;u=tail[rev[from[u]]])
            f=min(f,cap[from[u]]);
        for(u=t;u!=s;u=tail[rev[from[u]]])
        {
            cap[from[u]]-=f;
            cap[rev[from[u]]]+=f;
        }
    }
}

```

```

        flow+=f;  res+=f*h[t];
    }
    return P(flow,res);
}

```

2.8 Hungary

```

int nl,sz,first[N_LEFT],nxt[N_EDGE],tail[N_EDGE],vis[N_RIGHT],
    from[N_RIGHT];

// edges are linked from left to right only!
void addedge(int u,int v)
{
    tail[++sz]=v;
    nxt[sz]=first[u];  first[u]=sz;
}

int match(int u)
{
    int v,e;
    for(e=first[u];e=e=nxt[e])
    {
        v=tail[e];
        if(vis[v]) continue;
        vis[v]=1;
        if(!from[v] || match(from[v]))
        {
            from[v]=u;
            return 1;
        }
    }
    return 0;
}

int hungary()
{
    int ret=0,i;
    memset(from,0,sizeof(from));
    for(i=1;i<=nl;i++)
    {
        memset(vis,0,sizeof(vis));
        if(match(i)) ret++;
    }
    return ret;
}

```

2.9 Kuhn-Munkres

```

using LL=long long;
const LL INF=0x3f3f3f3f3f3f3f3f;
int n,from[N_NODE],vis[N_NODE*2];

```



```

LL lx[N_NODE],ly[N_NODE],edg[N_NODE][N_NODE],slack[N_NODE];

int match(int u,int o)
{
    vis[u]=1;
    int v;
    for(v=1;v<=n;v++)
        if(!vis[v+n] && edg[u][v]==lx[u]+ly[v])
        {
            vis[v+n]=1;
            if(!from[v] || match(from[v],o))
            {
                if(o) from[v]=u;
                return 1;
            }
        }
        else slack[v]=std::min(slack[v],lx[u]+ly[v]-edg[u][v]);
    return 0;
}

LL km()
{
    int i,j;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            lx[i]=std::max(lx[i],edg[i][j]);
    LL d;
    memset(from,0,sizeof(from));
    for(i=1;i<=n;i++)
    {
        memset(slack,0x3f,sizeof(slack));
        memset(vis,0,sizeof(vis));
        if(match(i,1)) continue;
        for(;;)
        {
            for(d=INF,j=1;j<=n;j++)
                if(!vis[j+n]) d=std::min(d,slack[j]);
            for(j=1;j<=n;j++)
                if(vis[j]) lx[j]-=d;
            for(j=1;j<=n;j++)
                if(vis[j+n]) ly[j]+=d;
            for(j=1;j<=n;j++)
                if(!vis[j+n]) slack[j]-=d;
            for(j=1;j<=n;j++)
                if(!vis[j+n] && !slack[j])
                {
                    vis[j+n]=1;
                    if(!from[j] || (!vis[from[j]] && match(from[j],0))) break;
                }
            if(j<=n) break;
        }
        memset(vis,0,sizeof(vis));
        match(i,1);
    }
    LL res=0;

```

```

    for(i=1;i<=n;i++) res+=lx[i];
    for(i=1;i<=n;i++) res+=ly[i];
    return res;
}

```

2.10 SCC decomposition

```

int n,sz,first[N_NODE],nxt[N_EDGE],tail[N_EDGE],cnt,l,sz2,f2[
    N_NODE],n2[N_EDGE],t2[N_EDGE],scc[N_NODE],s[N_NODE],stk[
    N_NODE],dfn[N_NODE],low[N_NODE];

void addedge(int u,int v)
{
    tail[++sz]=v;
    nxt[sz]=first[u];    first[u]=sz;
}

void addedge2(int u,int v)
{
    t2[++sz2]=v;
    n2[sz2]=f2[u];    f2[u]=sz2;
}

void dfs(int u)
{
    scc[u]=-1;
    low[u]=dfn[u]=++dfn[0];
    stk[++l]=u;
    int v,e;
    for(e=first[u];e;e=nxt[e])
    {
        v=tail[e];
        if(scc[v]>0) continue;
        if(!scc[v]) dfs(v);
        low[u]=std::min(low[u],low[v]);
    }
    if(low[u]==dfn[u])
    {
        s[++cnt]=0;
        while(stk[l]!=u)
        {
            scc[stk[l--]]=cnt;
            s[cnt]++;
        }
        scc[stk[l--]]=cnt;
        s[cnt]++;
    }
}

void scc_dec()
{
    memset(scc,0,sizeof(scc));
    int i;
    for(cnt=0,l=0,i=1;i<=n;i++)

```

```

        if(!scc[i]) dfs(i);
memset(f2,0,sizeof(f2));
int u,e;
for(sz2=0,u=1;u<=n;u++)
    for(e=first[u];e;e=nxt[e])
        if(scc[u]!=scc[tail[e]])
            addedge2(scc[u],scc[tail[e]]);
}

```

2.11 2-SAT

```

int solve()
{
    memset(dep,0,sizeof(dep));
    for(dfn[0]=scc[0]=cc[0]=0,lv=0,i=1;i<=m*2;i++)
        if(!dep[i])
        {
            ++cc[0];
            dep[i]=1;
            dfs(i);
        }
    for(i=1;i<=m;i++)
        if(scc[i]==scc[i+m]) return 0;
        else if(cc[i]<cc[i+m] || cc[i]==cc[i+m] && dep[i]>dep[i+m])
            chosen[i]=1;
        else chosen[i]=0;
    return 1;
}

void dfs(int u)
{
    low[u]=dfn[u]=++dfn[0];
    scc[u]=-1;
    cc[u]=cc[0];
    stk[++lv]=u;
    int v,e;
    for(e=first[u];e;e=nxt[e])
    {
        v=tail[e];
        if(!dep[v])
        {
            dep[v]=dep[u]+1;
            dfs(v);
            low[u]=min(low[u],low[v]);
        }
        else if(scc[v]==-1)
            low[u]=min(low[u],low[v]);
    }
    if(low[u]==dfn[u])
    {
        scc[0]++;
        while(stk[lv]!=u)
            scc[stk[lv--]]=scc[0];
        scc[stk[lv--]]=scc[0];
    }
}

```

```

    }
}

```

2.12 LCA (doubling)

```

LL sz, first[N_NODE], nxt[N_NODE*2], tail[N_NODE*2], len[N_NODE*2], p
[N_NODE][LOG_N_NODE], g[N_NODE][LOG_N_NODE], dep[N_NODE];

void addedge(LL u, LL v, LL w)
{
    sz++;
    tail[sz]=v; len[sz]=w;
    nxt[sz]=first[u]; first[u]=sz;
}

void dfs(LL u)
{
    LL v, e, i;
    for(i=1; i<=LOG_N; i++)
    {
        p[u][i]=p[p[u][i-1]][i-1];
        g[u][i]=max(g[u][i-1], g[p[u][i-1]][i-1]);
    }
    for(e=first[u]; e; e=nxt[e])
    {
        v=tail[e];
        if(dep[v]) continue;
        p[v][0]=u; g[v][0]=len[e];
        dep[v]=dep[u]+1;
        dfs(v);
    }
}

LL query(LL u, LL v)
{
    if(dep[u]>dep[v]) swap(u, v);
    LL i, ans=0;
    for(i=LOG_N; i>=0; i--)
        if(dep[v]-dep[u]>=(1<<i))
        {
            ans=max(ans, g[v][i]);
            v=p[v][i];
        }
    if(u==v) return ans;
    // return u;
    for(i=LOG_N; i>=0; i--)
        if(p[u][i]!=p[v][i])
        {
            ans=max(ans, max(g[u][i], g[v][i]));
            u=p[u][i]; v=p[v][i];
        }
    return max(ans, max(g[u][0], g[v][0]));
    // return p[u][0];
}

```

2.13 LCA (RMQ)

```
int n,sz,first[N_NODE],nxt[N_NODE*2],tail[N_NODE*2],dep[N_NODE],
    li[N_NODE],eul[N_NODE*2],b,minv[LOG_N_NODE][N_NODE*2];

void addedge(int u,int v)
{
    tail[++sz]=v;
    nxt[sz]=first[u];    first[u]=sz;
}

void dfs(int u)
{
    eul[++eul[0]]=u;      // initialize eul[0]
    li[u]=eul[0];        // initialize li[] to 0
    int v,e;
    for(e=first[u];e;e=nxt[e])
    {
        v=tail[e];
        if(li[v]) continue;
        dep[v]=dep[u]+1;
        dfs(v);
        eul[++eul[0]]=u;
    }
}

void rmq()
{
    b=sizeof(unsigned int)*8-__builtin_clz(n*2-1)-1;    // int
    __builtin_clz(unsigned int x)
    int i,j;
    for(i=1;i<=n*2-1;i++)
        minv[0][i]=eul[i];
    for(i=1;i<=b;i++)
        for(j=1;j<=n*2-1;j++)
            if(j+(1<<(i-1))>n*2-1 || dep[minv[i-1][j]]<dep[minv[i-1][j+(1<<(i-1))]])
                minv[i][j]=minv[i-1][j];
            else minv[i][j]=minv[i-1][j+(1<<(i-1))];
}

int query(int l,int r)
{
    int w=sizeof(unsigned int)*8-__builtin_clz(r-l+1)-1;
    if(dep[minv[w][l]]<dep[minv[w][r-(1<<w)+1]])
        return minv[w][l];
    else return minv[w][r-(1<<w)+1];
}

int lca(int u,int v)
{
    return query(std::min(li[u],li[v]),std::max(li[u],li[v]));
}
```

```
}
```

2.14 LCA (Tarjan)

```
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;
const int INF=1000000000;
struct PLAN
{
    int s,t,time,idx;
    bool operator<(const PLAN &rhs) const
    {
        return time<rhs.time;
    }
}
plan[N_QRY];
int n,m,nq,firstq[N_NODE],nxtq[N_QRY*2],to[N_QRY*2],idx[N_QRY*2],lca[N_QRY],sz,first[N_NODE],nxt[N_NODE*2],tail[N_NODE*2],len[N_NODE*2],vis[N_NODE],f[N_NODE];

void addedge(int u,int v,int l)
{
    sz++;
    tail[sz]=v; len[sz]=l;
    nxt[sz]=first[u]; first[u]=sz;
}

void addqry(int u,int v,int i)
{
    nq++;
    to[nq]=v; idx[nq]=i;
    nxtq[nq]=firstq[u]; firstq[u]=nq;
}

int find(int u)
{
    return (f[u]==u ? u : (f[u]=find(f[u])));
}

void get_lca(int u)
{
    vis[u]=1;
    int q,v,e;
    for(q=firstq[u];q;q=nxtq[q])
    {
        v=to[q];
        if(!vis[v]) continue;
        lca[idx[q]]=find(v);
    }
    for(e=first[u];e;e=nxt[e])
    {
```

```

        v=tail[e];
        if(vis[v]) continue;
        get_lca(v);
        f[v]=u;
    }
}

int main()
{
    scanf("%d%d",&n,&m);
    memset(first,0,sizeof(first));
    int i,u,v,l;
    for(sz=0,i=1;i<=n-1;i++)
    {
        scanf("%d%d%d",&u,&v,&l);
        addedge(u,v,l);    addedge(v,u,l);
    }
    memset(firstq,0,sizeof(firstq));
    for(nq=0,i=1;i<=m;i++)
    {
        scanf("%d%d",&u,&v);
        plan[i].s=u;    plan[i].t=v;
        plan[i].idx=i;
        addqry(u,v,i);    addqry(v,u,i);
    }
    for(i=1;i<=n;i++) f[i]=i;
    memset(vis,0,sizeof(vis));
    get_lca(1);    // 1 is the root
    return 0;
}

```

2.15 Diameter of tree

```

int n,sz,first[N_NODE],nxt[N_NODE*2],tail[N_NODE*2],dep[N_NODE],
par[N_NODE],diam[N_NODE];

void dfs(int u)
{
    int v,e;
    for(e=first[u];e;e=nxt[e])
    {
        v=tail[e];
        if(dep[v]) continue;
        par[v]=u;
        dep[v]=dep[u]+1;
        dfs(v);
    }
}

int diameter(int &rt)    // return length of tree's diameter
{
    int i;
    memset(dep,0,sizeof(dep));
    dep[1]=1;
}

```

```

    dfs(1);
    for(rt=0,i=1;i<=n;i++)
        if(dep[i]>dep[rt]) rt=i;
    memset(dep,0,sizeof(dep));
    dep[rt]=1;
    dfs(rt);
    int t=0;
    for(i=1;i<=n;i++)
        if(dep[i]>dep[t]) t=i;
    int u,len=0;
    for(u=t;u!=rt;u=par[u])
        diam[++len]=u;
    diam[++len]=rt;
    return len;
}

```

2.16 Maximum density subgraph

```

#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>

const double INF=1e5,EPS=1e-18,LIM=1e-5;
int n,m,cv,ce,first[2000],nxt[8000],tail[8000],rev[8000],level[2000],cur[2000],que[2000],ans[110];
double cap[8000],cap0[8000];
void addedge(int u,int v,double c);
double dinic(int s,int t);
void bfs(int s);
double dfs(int u,int t,double f);

int main()
{
    freopen("life.in","r",stdin);
    freopen("life.out","w",stdout);
    scanf("%d%d",&n,&m);
    memset(first,0,sizeof(first));
    int i,u,v;
    for(ce=0,i=1;i<=m;i++)
    {
        scanf("%d%d",&u,&v);
        addedge(i,u+m,INF); addedge(i,v+m,INF);
    }
    int s=n+m+1,t=n+m+2;
    for(i=1;i<=m;i++) addedge(s,i,1);
    for(i=1;i<=n;i++) addedge(i+m,t,0);
    for(i=1;i<=ce;i++) cap0[i]=cap[i];
    cv=n+m+2;
    double l=0,r=m,mid;
    while(r-l>LIM)
    {
        mid=(l+r)/2;
        for(i=1;i<=ce;i++)

```



```

        if(tail[i]==t) cap[i]=mid;
        else cap[i]=cap0[i];
    if(m-dinic(s,t)>EPS)
    {
        l=mid;
        for(ans[0]=0,i=1;i<=n;i++)
            if(level[i+m]>=0)
                ans[++ans[0]]=i;
    }
    else r=mid;
}
std::sort(ans+1,ans+ans[0]+1);
if(!ans[0]) ans[ans[0]=1]=1;
for(i=0;i<=ans[0];i++)
    printf("%d\n",ans[i]);
return 0;
}

void addedge(int u,int v,double c)
{
    ce++;
    tail[ce]=v; cap[ce]=c;
    nxt[ce]=first[u]; first[u]=ce;
    ce++;
    tail[ce]=u; cap[ce]=0;
    nxt[ce]=first[v]; first[v]=ce;
    rev[ce]=ce-1; rev[ce-1]=ce;
}

double dinic(int s,int t)
{
    double ans=0,f;
    int i;
    for(;;)
    {
        bfs(s);
        if(level[t]<0) break;
        for(i=1;i<=cv;i++)
            cur[i]=first[i];
        while((f=dfs(s,t,INF))>0) ans+=f;
    }
    return ans;
}

void bfs(int s)
{
    memset(level,-1,sizeof(level));
    level[s]=0;
    int front=1,rear=1,u,v,e;
    que[1]=s;
    while(front<=rear)
    {
        u=que[front++];
        for(e=first[u];e;e=nxt[e])
        {
            v=tail[e];

```

```

        if(fabs(cap[e])<EPS || level[v]>=0) continue;
        level[v]=level[u]+1;
        que[++rear]=v;
    }
}

double dfs(int u,int t,double f)
{
    if(u==t) return f;
    int v,&e=cur[u];
    double d;
    for(;e;e=nxt[e])
    {
        v=tail[e];
        if(fabs(cap[e])<EPS || level[u]>=level[v]) continue;
        d=dfs(v,t,std::min(f, cap[e]));
        if(d>0)
        {
            cap[e]-=d;
            cap[rev[e]]+=d;
            return d;
        }
    }
    return 0;
}

```

3 Data structures

3.1 Discretization

```

int n,l[N_SEG],r[N_SEG],l2[N_SEG],r2[N_SEG],x[N_SEG*2];

void discrete()
{
    int i,j;
    for(i=1;i<=n;i++)
        x[2*i-1]=l[i],x[2*i]=r[i];
    std::sort(x+1,x+2*n+1);
    int *arr[2]={l,r},*arrn[2]={l2,r2},lb,ub,mid;
    for(i=1;i<=n;i++)
        for(j=0;j<=1;j++)
        {
            lb=1; ub=n*2;
            while(lb<ub)
            {
                mid=(lb+ub)/2;
                if(arr[j][i]<=x[mid]) ub=mid;
                else lb=mid+1;
            }
            arrn[j][i]=lb;
        }
}

```

3.2 Segment tree (add)

```
void update(int p,int q,LL val,int k,int l,int r)
{
    if(p<=l && q>=r)
    {
        add[k]+=val;
        maintain(k,l,r);
        return;
    }
    int mid=(l+r)/2;
    if(p<=mid) update(p,q,val,k*2,l,mid);
    if(q>mid) update(p,q,val,k*2+1,mid+1,r);
    maintain(k,l,r);
}

void query(int p,int q,int k,int l,int r,LL s,LL &res)
{
    if(p<=l && q>=r)
    {
        res+=sum[k]+s*(r-l+1);
        return;
    }
    int mid=(l+r)/2;
    if(p<=mid) query(p,q,k*2,l,mid,s+add[k],res);
    if(q>mid) query(p,q,k*2+1,mid+1,r,s+add[k],res);
}

void maintain(int k,int l,int r)
{
    if(l==r) sum[k]=add[k];
    else sum[k]=sum[k*2]+sum[k*2+1]+add[k]*(r-l+1);
}
```

3.3 Segment tree (add, set)

```
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;
typedef long long LL;
const LL INF=1000000000000000;
LL n,len,q,a[1000010],tag[2][2000010],sum[2000010],mx[2000010],
mn[2000010],ans[3];
void build(LL k,LL l,LL r);
void update(LL op,LL p,LL q,LL val,LL k,LL l,LL r);
void query(LL p,LL q,LL k,LL l,LL r,LL add,LL &res_s,LL &res_a,
LL &res_i);
void maintain(LL k,LL l,LL r);
```

```

void pushdown(LL k,LL l,LL r);

int main()
{
    freopen("segment.in","r",stdin);
    freopen("segment.out","w",stdout);
    scanf("%lld",&n);
    memset(a,0,sizeof(a));
    LL i;
    for(i=1;i<=n;i++)
        scanf("%lld",&a[i]);
    for(len=1;len<n;len*=2);
    build(1,1,len);
    scanf("%lld",&q);
    LL op,l,r,x;
    while(q--)
    {
        scanf("%lld",&op);
        if(op==1)
        {
            scanf("%lld%lld%lld",&l,&r,&x);
            update(1,l,r,x,1,1,len);
        }
        else if(op==2)
        {
            scanf("%lld%lld%lld",&l,&r,&x);
            update(0,l,r,x,1,1,len);
        }
        else
        {
            scanf("%lld%lld",&l,&r);
            ans[0]=0,ans[1]=-INF,ans[2]=INF;
            query(1,r,1,1,len,0,ans[0],ans[1],ans[2]);
            printf("%lld□%lld□%lld\n",ans[0],ans[1],ans[2]);
        }
    }
    return 0;
}

void build(LL k,LL l,LL r)
{
    if(l==r)
    {
        tag[0][k]=a[l];    tag[1][k]=0;
        maintain(k,l,r);
        return;
    }
    LL mid=(l+r)/2;
    build(k*2,l,mid);    build(k*2+1,mid+1,r);
    tag[0][k]=INF;    tag[1][k]=0;
    maintain(k,l,r);
}

void update(LL op,LL p,LL q,LL val,LL k,LL l,LL r)
{
    if(p<=l && q>=r)

```

```

{
    if(op) tag[1][k] += val;
    else tag[0][k] = val, tag[1][k] = 0;
    maintain(k, l, r);
    return;
}
pushdown(k, l, r);
LL mid = (l + r) / 2;
if(p <= mid) update(op, p, q, val, k * 2, l, mid);
if(q > mid) update(op, p, q, val, k * 2 + 1, mid + 1, r);
maintain(k, l, r);
}

void query(LL p, LL q, LL k, LL l, LL r, LL add, LL &res_s, LL &res_a,
LL &res_i)
{
    if(p <= l && q >= r)
    {
        res_s += sum[k] + add * (r - l + 1);
        res_a = max(res_a, mx[k] + add); res_i = min(res_i, mn[k] + add);
        return;
    }
    else if(tag[0][k] != INF)
    {
        res_s += (tag[0][k] + tag[1][k] + add) * (min(q, r) - max(p, l) + 1);
        res_a = max(res_a, tag[0][k] + tag[1][k] + add);
        res_i = min(res_i, tag[0][k] + tag[1][k] + add);
        return;
    }
    LL mid = (l + r) / 2;
    if(p <= mid) query(p, q, k * 2, l, mid, add + tag[1][k], res_s, res_a,
res_i);
    if(q > mid) query(p, q, k * 2 + 1, mid + 1, r, add + tag[1][k], res_s, res_a,
res_i);
}

void maintain(LL k, LL l, LL r)
{
    if(tag[0][k] != INF)
    {
        sum[k] = (tag[0][k] + tag[1][k]) * (r - l + 1);
        mx[k] = mn[k] = tag[0][k] + tag[1][k];
    }
    else
    {
        sum[k] = sum[k * 2] + sum[k * 2 + 1] + tag[1][k] * (r - l + 1);
        mx[k] = max(mx[k * 2], mx[k * 2 + 1]) + tag[1][k];
        mn[k] = min(mn[k * 2], mn[k * 2 + 1]) + tag[1][k];
    }
}

void pushdown(LL k, LL l, LL r)
{
    if(tag[0][k] != INF)
    {
        tag[0][k * 2] = tag[0][k * 2 + 1] = tag[0][k];

```

```

        tag[1][k*2]=tag[1][k*2+1]=tag[1][k];
    }
    else
    {
        tag[1][k*2]+=tag[1][k];    tag[1][k*2+1]+=tag[1][k];
    }
    LL mid=(l+r)/2;
    maintain(k*2,l,mid);    maintain(k*2+1,mid+1,r);
    tag[0][k]=INF;    tag[1][k]=0;
    maintain(k,l,r);
}

```

3.4 Segment tree (add, multiply)

```

int n,m,son[100010],bro[100010],cc,s[100010],par[100010],idx
[100010],belong[100010],top[100010],mulv[400010],addv
[400010],sum[400010];

void maintain(int k,int l,int r)
{
    if(l==r) sum[k]=addv[k];
    else sum[k]=(sum[k*2]+sum[k*2+1])*mulv[k]+addv[k]*(r-l+1);
}

void pushdown(int k,int l,int r)
{
    int mid=(l+r)/2;
    mulv[k*2]*=mulv[k];    addv[k*2]=addv[k*2]*mulv[k]+addv[k];
    maintain(k*2,l,mid);
    mulv[k*2+1]*=mulv[k];    addv[k*2+1]=addv[k*2+1]*mulv[k]+addv[
        k];
    maintain(k*2+1,mid+1,r);
    mulv[k]=1;    addv[k]=0;
    maintain(k,l,r);
}

void build(int k,int l,int r)
{
    mulv[k]=1;    addv[k]=0;
    if(l<r)
    {
        int mid=(l+r)/2;
        build(k*2,l,mid);    build(k*2+1,mid+1,r);
    }
    maintain(k,l,r);
}

void update(int p,int q,int val,int o,int k,int l,int r)    // o
=0: multiply; o=1: add
{
    if(p<=l && q>=r)
    {
        if(!o) mulv[k]*=val,addv[k]*=val;
        else addv[k]+=val;
    }
}

```

```

        maintain(k,l,r);
        return;
    }
    pushdown(k,l,r);
    int mid=(l+r)/2;
    if(p<=mid) update(p,q,val,o,k*2,l,mid);
    if(q>mid) update(p,q,val,o,k*2+1,mid+1,r);
    maintain(k,l,r);
}

int query(int p,int q,int k,int l,int r,int mul,int add)
{
    if(p<=l && q>=r)
        return sum[k]*mul+add*(r-l+1);
    int mid=(l+r)/2,res=0;
    add=mul*addv[k]+add; mul*=mulv[k];
    if(p<=mid) res+=query(p,q,k*2,l,mid,mul,add);
    if(q>mid) res+=query(p,q,k*2+1,mid+1,r,mul,add);
    return res;
}

```

3.5 Segment tree (sum of sums)

```

void build(int k,int l,int r)
{
    if(l==r)
    {
        ls[k]=rs[k]=ss[k]=sum[k]=a[l];
        return;
    }
    int mid=(l+r)/2;
    build(k*2,l,mid); build(k*2+1,mid+1,r);
    sum[k]=(sum[k*2]+sum[k*2+1])%M;
    ls[k]=(ls[k*2]+ls[k*2+1]+sum[k*2]*(r-mid)%M)%M;
    rs[k]=(rs[k*2]+rs[k*2+1]+sum[k*2+1]*(mid-l+1)%M)%M;
    ss[k]=(ss[k*2]+ss[k*2+1]+rs[k*2]*(r-mid)%M+ls[k*2+1]*(mid-l+1)%M)%M;
}

LL query(int p,int q,int k,int l,int r)
{
    if(p<=l && q>=r)
        return (ss[k]+sum[k]*(l-p)%M*(q-r)%M+ls[k]*(l-p)%M+rs[k]*(q-r)%M)%M;
    int mid=(l+r)/2;
    LL res=0;
    if(p<=mid) res=(res+query(p,q,k*2,l,mid))%M;
    if(q>mid) res=(res+query(p,q,k*2+1,mid+1,r))%M;
    return res;
}

```

3.6 Heap

```
struct HEAPNODE
{
    int val,idx;
};

void up(int u)
{
    if(u>1 && heap[u].val>heap[u/2].val)
    {
        std::swap(heap[u],heap[u/2]);
        up(u/2);
    }
}

void down(int u)
{
    if(u*2>h) return;
    int v=u*2;
    if(v+1<=h && heap[v+1].val>heap[v].val) v++;
    if(heap[v].val>heap[u].val)
    {
        std::swap(heap[u],heap[v]);
        down(v);
    }
}

void push(int val,int idx)
{
    h++;
    heap[h].val=val; heap[h].idx=idx;
    up(h);
}

HEAPNODE pop()
{
    HEAPNODE res=heap[1];
    std::swap(heap[1],heap[h--]);
    down(1);
    return res;
}
```

3.7 RMQ

```
int n,b,arr[SIZE],minv[LOG_SIZE][SIZE*2];

void rmq()
{
    b=sizeof(unsigned int)*8-__builtin_clz(n)-1;           // int
    __builtin_clz(unsigned int x)
    memset(minv,0x3f,sizeof(minv));
    int i,j;
```



```

    for(i=1;i<=n;i++)
        minv[0][i]=arr[i];
    for(i=1;i<=b;i++)
        for(j=1;j<=n;j++)
            minv[i][j]=std::min(minv[i-1][j],minv[i-1][j+(1<<(i-1))]);
}

int query(int l,int r)
{
    int w=sizeof(unsigned int)*8-__builtin_clz(r-l+1)-1;
    return std::min(minv[w][l],minv[w][r-(1<<w)+1]);
}

```

3.8 Weighted union find

```

int n,k,f[N_NODE],d[N_NODE]; // initialize f[i] to i, d[i] to 0

int find(int u)
{
    if(f[u]==u) return u;
    int t=find(f[u]);
    d[u]=(d[u]+d[f[u]])%MOD;
    f[u]=t;
    return t;
}

int dist(int u)
{
    find(u);
    return d[u];
}

int link(int u,int v,int l) // link u to v (l is the
    distance from u to v)
{
    int fu=find(u),fv=find(v);
    if(fu==fv) return 0; // return 0 if u and v have been
        in the same set
    f[fu]=fv;
    d[fu]=(l+d[v]-d[u]+MOD)%MOD;
    return 1;
}

```

3.9 Treap

```

const unsigned SEED=19260817;
int sz,ls[N_NODE],rs[N_NODE],par[N_NODE],key[N_NODE],val[N_NODE],
    hk[N_NODE],cnt[N_NODE];

void maintain(int u)

```

```

{
    cnt[u]=cnt[ls[u]]+cnt[rs[u]]+1;
}

int lrot(int u)
{
    int v=rs[u];
    rs[u]=ls[v];  ls[v]=u;
    par[v]=par[u];  par[u]=v;  par[rs[u]]=u;
    maintain(u);  maintain(v);
    return v;
}

int rrot(int u)
{
    int v=ls[u];
    ls[u]=rs[v];  rs[v]=u;
    par[v]=par[u];  par[u]=v;  par[ls[u]]=u;
    maintain(u);  maintain(v);
    return v;
}

int insert(int u,int p,int kk,int vv)
{
    if(!u)
    {
        sz++;
        par[sz]=p;  ls[sz]=rs[sz]=0;
        key[sz]=kk;  val[sz]=vv;
        hk[sz]=rand()+1;
        cnt[sz]=1;
        return sz;
    }
    if(kk<key[u])
        ls[u]=insert(ls[u],u,kk,vv);
    else rs[u]=insert(rs[u],u,kk,vv);
    maintain(u);
    if(hk[ls[u]]>hk[u]) u=rrot(u);
    else if(hk[rs[u]]>hk[u]) u=lrot(u);
    return u;
}

int erase(int u,int targ)
{
    if(u==targ)
    {
        if(!ls[u] && !rs[u])
        {
            par[u]=0;
            return 0;
        }
        if(hk[ls[u]]>hk[rs[u]])
        {
            u=rrot(u);
            rs[u]=erase(rs[u],targ);
        }
    }
}

```

```

        else
        {
            u=lrot(u);
            ls[u]=erase(ls[u],targ);
        }
        maintain(u);
        return u;
    }
    if(key[targ]<key[u])
        ls[u]=erase(ls[u],targ);
    else rs[u]=erase(rs[u],targ);
    maintain(u);
    return u;
}

int newnode(int kk,int vv)
{
    sz++;
    par[sz]=ls[sz]=rs[sz]=0;
    key[sz]=kk; val[sz]=vv;
    hk[sz]=rand()+1;
    cnt[sz]=1;
    return sz;
}

int insert_node(int u,int p,int v)
{
    if(!u)
    {
        par[v]=p;
        return v;
    }
    if(key[v]<key[u])
        ls[u]=insert_node(ls[u],u,v);
    else rs[u]=insert_node(rs[u],u,v);
    maintain(u);
    if(hk[ls[u]]>hk[u]) u=rrot(u);
    else if(hk[rs[u]]>hk[u]) u=lrot(u);
    return u;
}

int find(int u,int kk)
{
    if(!u) return 0;
    else if(kk==key[u]) return u;
    else if(kk<key[u])
        return find(ls[u],kk);
    else return find(rs[u],kk);
}

int pred(int u)
{
    if(ls[u])
    {
        int v=ls[u];
        while(rs[v]) v=rs[v];
    }
}

```

```

        return v;
    }
    int p=u,q=par[u];
    while(q && p==ls[q])
        p=q,q=par[q];
    return q;
}

int succ(int u)
{
    if(rs[u])
    {
        int v=rs[u];
        while(ls[v]) v=ls[v];
        return v;
    }
    int p=u,q=par[u];
    while(q && p==rs[q])
        p=q,q=par[q];
    return q;
}

int rank(int u,int kk)
{
    if(!u) return 0;
    else if(kk==key[u]) return cnt[ls[u]];
    else if(kk<key[u])
        return rank(ls[u],kk);
    else return rank(rs[u],kk)+cnt[ls[u]]+1;
}

int select(int u,int k)
{
    int t=cnt[ls[u]];
    if(k==t+1) return u;
    else if(k<t+1)
        return select(ls[u],k);
    else return select(rs[u],k-cnt[ls[u]]-1);
}

void merge(int &dest,int &src)
{
    int t;
    while(src)
    {
        t=src;
        src=erase(src,src);
        dest=insert_node(dest,0,t);
    }
}

```

3.10 Cartesian tree

```

int n,ls[N_NODE],rs[N_NODE],stk[N_NODE];

int carttree(int *a)
{
    int i,m=0,rt=0;
    for(i=1;i<=n;i++)
    {
        while(m && a[i]>a[stk[m]]) m--;           // the largest
                                                value is the root
        ls[i]=m ? rs[stk[m]] : rt;
        rs[i]=0;
        if(m) rs[stk[m]]=i;
        else rt=i;
        stk[++m]=i;
    }
    return rt;
}

```

3.11 Expression tree

```

int build_expr(char *str)    // return root of expression tree
{
    int numn=0,opn=0,l=strlen(str),i;
    for(i=0;i<l;i++)
    {
        if(str[i]>='0' && str[i]<='9') nums[++numn]='0'-str[i];
        // leaves(operands) have negative tree-node
        indices
        else if(str[i]=='(' || str[i]=='?') ops[++opn]=str[i];
        else
        {
            while(opn && ops[opn]!='(')
            {
                sz++;
                rs[sz]=nums[numn--];  ls[sz]=nums[numn--];
                s[sz]=(ls[sz]>0 ? s[ls[sz]] : 0)+(rs[sz]>0 ? s[
                    rs[sz]] : 0)+1;
                nums[++numn]=sz;
                opn--;
            }
            if(opn) opn--;
        }
    }
    return nums[1];
}

```

3.12 Persistent segment tree

```

// updates for single positions, and querys for prefix sums (
// similar to BIT)

```

```

LL n,a[100010],sz,ls[3000000],rs[3000000],sum[3000000],rt
[100010];

LL build(LL l,LL r)
{
    LL u=++sz;          // sz needs to be initialized
    sum[u]=0;
    if(l<r)
    {
        LL m=(l+r)/2;
        ls[u]=build(l,m);  rs[u]=build(m+1,r);
    }
    return u;
}

LL update(LL pos,LL val,LL u,LL l,LL r)
{
    if(l==r)
    {
        sum[++sz]=sum[u]+val;
        return sz;
    }
    LL v=++sz,m=(l+r)/2;
    if(pos<=m)
    {
        ls[v]=update(pos,val,ls[u],l,m);
        rs[v]=rs[u];
    }
    else
    {
        ls[v]=ls[u];
        rs[v]=update(pos,val,rs[u],m+1,r);
    }
    sum[v]=sum[ls[v]]+sum[rs[v]];
    return v;
}

LL query(LL pos,LL u,LL l,LL r)
{
    if(pos<l) return 0;
    else if(pos==r) return sum[u];
    LL m=(l+r)/2;
    if(pos>m) return sum[ls[u]]+query(pos,rs[u],m+1,r);
    else return query(pos,ls[u],l,m);
}

int main()
{
    scanf("%lld",&n);
    LL i;
    for(i=1;i<=n;i++)
        scanf("%lld",&a[i]);
    sz=0;
    rt[0]=build(1,n);
    for(i=1;i<=n;i++)
        rt[i]=update(a[i],1,rt[i-1],1,n);
}

```

```

    return 0;
}

```

3.13 Chain decomposition

```

int cc,s[N_NODE],par[N_NODE],idx[N_NODE],belong[N_NODE],top[
N_NODE];

void dfs(int u)
{
    int v,e;
    for(s[u]=1,e=first[u];e;nxt[e])
    {
        v=tail[e];
        if(v==par[u]) continue;           // par[root] should be
            initialized
        par[v]=u;
        dfs(v);
        s[u]+=s[v];
    }
}

void split(int u)
{
    idx[u]=++idx[0];    // idx[0] should be initialized
    int v,e,v0=0;
    for(e=first[u];e;nxt[e])
    {
        v=tail[e];
        if(par[v]!=u) continue;
        if(s[v]>s[v0]) v0=v;
    }
    if(!v0)
    {
        belong[u]=++cc;    // cc should be initialized
        top[cc]=u;
        // ridx[u]=idx[0];
        return;
    }
    split(v0);
    belong[u]=belong[v0];
    top[belong[u]]=u;
    for(e=first[u];e;nxt[e])
    {
        v=tail[e];
        if(par[v]!=u || v==v0) continue;
        split(v);
    }
    // ridx[u]=idx[0];
}

int lca(int u,int v)
{
    while(belong[u]!=belong[v])

```

```

    {
        if(idx[top[belong[u]]]>idx[top[belong[v]]]) std::swap(u,
            v);
        v=par[top[belong[v]]];
    }
    return idx[u]<idx[v] ? u : v;
}

void update2(int u,int v,int val)
{
    while(belong[u]!=belong[v])
    {
        if(idx[top[belong[u]]]>idx[top[belong[v]]]) std::swap(u,
            v);
        // update(idx[top[belong[v]]],idx[v],val,1,1,n);
        v=par[top[belong[v]]];
    }
    if(idx[u]>idx[v]) std::swap(u,v);
    // update(idx[u],idx[v],val,1,1,n);
}

int query2(int u,int v)
{
    int res=0;
    while(belong[u]!=belong[v])
    {
        if(idx[top[belong[u]]]>idx[top[belong[v]]]) std::swap(u,
            v);
        // res+=query(idx[top[belong[v]]],idx[v],1,1,n);
        v=par[top[belong[v]]];
    }
    if(idx[u]>idx[v]) std::swap(u,v);
    // res+=query(idx[u],idx[v],1,1,n);
    return res;
}

```

4 String

4.1 KMP

```

int n,m,fail[LEN_PATTERN],match[LEN],trans[LEN_PATTERN][ALPHABET
];

void kmp(int *s1,int *s2)
{
    int i,j;
    for(fail[0]=fail[1]=0,i=2;i<=m;i++)
    {
        for(j=fail[i-1];j;j=fail[j])
            if(s2[j+1]==s2[i])
            {
                fail[i]=j+1;
                break;
            }
    }
}

```



```

    }
    if(!j) fail[i]=(s2[1]==s2[i] ? 1 : 0);
}
memset(trans[0],0,sizeof(trans[0]));
trans[0][s2[1]]=1;
for(i=1;i<=m;i++)
    for(j=1;j<=ALPHABET;j++)
        trans[i][j]=(i<m && s2[i+1]==j) ? i+1 : trans[fail[i]]
            [j];
for(match[0]=0,i=1;i<=n;i++)
{
    for(j=match[i-1];j;j=fail[j])
        if(j+1<=m && s2[j+1]==s1[i])
        {
            match[i]=j+1;
            break;
        }
    if(!j) match[i]=(s2[1]==s1[i] ? 1 : 0);

    match[i]=trans[match[i-1]][s1[i]];        // with array
    trans[i][j]
}
}

```

4.2 Aho-Corasick automaton

```

char str[N_STRING][MAX_LEN];
int n,sz,son[N_NODE][ALPHABET],par[N_NODE],alp[N_NODE],fail[
N_NODE],que[N_NODE],term[N_NODE],mat[LENGTH],trans[N_NODE][
ALPHABET];

void build_trie()
{
    int i,j,u;
    memset(son[1],0,sizeof(son[1]));
    term[1]=0;
    for(sz=1,i=1;i<=n;i++)
    {
        for(u=1,j=0;str[i][j];j++)
        {
            if(!son[u][str[i][j]-'a'])        // lowercase only
            {
                son[u][str[i][j]-'a']=++sz;
                memset(son[sz],0,sizeof(son[sz]));
                par[sz]=u;
                alp[sz]=str[i][j]-'a';
                term[sz]=0;
            }
            u=son[u][str[i][j]-'a'];
        }
        term[u]=1;
    }
}

```

```

void build_ac()
{
    int front=1, rear=1, u, v, j;
    que[1]=1;
    while(front<=rear)
    {
        u=que[front++];
        if(u==1 || par[u]==1) fail[u]=1;
        else
        {
            for(v=fail[par[u]]; v>1; v=fail[v])
                if(son[v][alp[u]])
                {
                    fail[u]=son[v][alp[u]];
                    break;
                }
            if(v==1) fail[u]=(son[1][alp[u]] ? son[1][alp[u]] : 1);
        }
        term[u]=term[fail[u]];
        for(j=0; j<26; j++)
            if(u==1) trans[u][j]=(son[u][j] ? son[u][j] : 1);
            else trans[u][j]=(son[u][j] ? son[u][j] : trans[fail[u]][j]);
        for(j=0; j<26; j++) // lowercase only
            if(son[u][j]) que[++rear]=son[u][j];
    }
}

void match(char *str)
{
    int i, u, v;
    char x;
    for(u=1, i=0; str[i]; i++)
    {
        x=str[i]-'a';

        for(v=u; v>1; v=fail[v])
            if(son[v][x])
            {
                mat[i]=son[v][x];
                break;
            }
        if(v==1) mat[i]=(son[1][x] ? son[1][x] : 1);

        mat[i]=trans[u][x]; // with array trans[][]

        u=mat[i];
    }
}

```

4.3 Suffix array

```

char a[LENGTH];
int n,ofs,sa[LENGTH],hei[LENGTH],rk[LENGTH],tmp[LENGTH];

// radix sort
int c[ALPHABET];
void rsort() // rk[]>0
{
    int b,i;
    for(b=1;b>=0;b--)
    {
        memset(c,0,sizeof(c));
        for(i=1;i<=n;i++)
            c[sa[i]+ofs*b<=n ? rk[sa[i]+ofs*b] : 0]++;
        for(i=1;i<=MAX_RANK;i++) c[i]+=c[i-1];
        for(i=n;i>=1;i--)
            tmp[c[sa[i]+ofs*b<=n ? rk[sa[i]+ofs*b] : 0]--]=sa[i];
        for(i=1;i<=n;i++) sa[i]=tmp[i];
    }
}

bool cmp(int i,int j)
{
    if(rk[i]!=rk[j]) return rk[i]<rk[j];
    int ri=(i+ofs<=n ? rk[i+ofs] : -1),rj=(j+ofs<=n ? rk[j+ofs] : -1); // rank>=0
    return ri<rj;
}

void build_sa()
{
    int i;
    for(i=1;i<=n;i++)
    {
        sa[i]=i;
        rk[i]=a[i];
    }
    for(ofs=1;ofs<=n;ofs*=2)
    {
        sort(sa+1,sa+n+1,cmp);

        // rsort();
        // radix sort -- construct suffix array in O(nlogn)

        for(tmp[sa[1]]=1,i=2;i<=n;i++)
            tmp[sa[i]]=tmp[sa[i-1]]+(cmp(sa[i-1],sa[i]) ? 1 : 0);
        for(i=1;i<=n;i++) rk[i]=tmp[i];
    }
}

void build_hei()
{
    int h=0,i,j;
    for(i=1;i<=n;i++)
        rk[sa[i]]=i;

```

```

    for(i=1;i<=n;i++)
    {
        if(rk[i]==n)
        {
            hei[n]=0;
            continue;
        }
        j=sa[rk[i]+1];
        if(h) h--;
        for(;j+h<=n && i+h<=n;h++)
            if(a[j+h]!=a[i+h]) break;
        hei[rk[i]]=h;
    }
}

/* Count the number of distinct substrings:

#include <stdio>
#include <string>
#include <algorithm>

typedef long long LL;
char a[100010];
int n,ofs,sa[100010],hei[100010],rk[100010],tmp[100010],l
    [100010],r[100010],stk[100010],last[100010];

// radix sort
int c[100010];
void rsort() // rk[]>0
{
    int b,i;
    for(b=1;b>=0;b--)
    {
        memset(c,0,sizeof(c));
        for(i=1;i<=n;i++)
            c[sa[i]+ofs*b<=n ? rk[sa[i]+ofs*b] : 0]++;
        for(i=1;i<=std::max(n,200);i++) c[i]+=c[i-1];
        for(i=n;i>=1;i--)
            tmp[c[sa[i]+ofs*b<=n ? rk[sa[i]+ofs*b] : 0]--]=sa[i]
                ];
        for(i=1;i<=n;i++) sa[i]=tmp[i];
    }
}

bool cmp(int i,int j)
{
    if(rk[i]!=rk[j]) return rk[i]<rk[j];
    int ri=(i+ofs<=n ? rk[i+ofs] : -1),rj=(j+ofs<=n ? rk[j+ofs]
        : -1); // rank>=0
    return ri<rj;
}

void build_sa()
{
    int i;
    for(i=1;i<=n;i++)

```

```

{
    sa[i]=i;
    rk[i]=a[i];
}
for(ofs=1;ofs<=n;ofs*=2)
{
    rsort();
    for(tmp[sa[1]]=1,i=2;i<=n;i++)
        tmp[sa[i]]=tmp[sa[i-1]]+(cmp(sa[i-1],sa[i]) ? 1 : 0)
        ;
    for(i=1;i<=n;i++) rk[i]=tmp[i];
}
}

void build_hei()
{
    int h=0,i,j;
    for(i=1;i<=n;i++)
        rk[sa[i]]=i;
    for(i=1;i<=n;i++)
    {
        if(rk[i]==n)
        {
            hei[n]=0;
            continue;
        }
        j=sa[rk[i]+1];
        if(h) h--;
        for(;j+h<=n && i+h<=n;h++)
            if(a[j+h]!=a[i+h]) break;
        hei[rk[i]]=h;
    }
}

int main()
{
    scanf("%d%s",&n,a+1);
    build_sa(); build_hei();
    int i,j;
    for(j=0,i=1;i<=n;i++)
    {
        while(j && hei[i]<=hei[stk[j]]) j--;
        l[i]=j ? stk[j] : 0;
        stk[++j]=i;
    }
    for(j=0,i=n;i>=1;i--)
    {
        while(j && hei[i]<=hei[stk[j]]) j--;
        r[i]=j ? stk[j] : n+1;
        stk[++j]=i;
    }
    memset(last,-1,sizeof(last));
    LL ans=0;
    for(hei[0]=hei[n+1]=0,i=1;i<=n;i++)
    {
        ans+=n-sa[i]+1-std::max(hei[i-1],hei[i]);
    }
}

```

```

        if(l[i]>last[hei[i]])
            ans+=hei[i]-std::max(hei[l[i]],hei[r[i]]);
        // cnt[i]=r[i]-l[i];
        last[hei[i]]=i;
    }
    printf("%lld\n",ans);
    return 0;
}
*/

```

4.4 Suffix array (SA-IS)

```

#include <cstdio>
#include <cstring>
#include <algorithm>

char str[LENGTH];
int n,a[LENGTH*2],sa[LENGTH*2],typ[LENGTH*2],c[LENGTH+ALPHABET],
    p[LENGTH],sbuc[LENGTH+ALPHABET],lbuc[LENGTH+ALPHABET],name[
    LENGTH],hei[LENGTH],rk[LENGTH];

inline int islms(int *typ,int i)
{
    return !typ[i] && (i==1 || typ[i-1]);
}

int cmp(int *s,int *typ,int p,int q)
{
    do
    {
        if(s[p]!=s[q]) return 1;
        p++; q++;
    }
    while(!islms(typ,p) && !islms(typ,q));
    return (!islms(typ,p) || !islms(typ,q) || s[p]!=s[q]);
}

void isort(int *s,int *sa,int *typ,int *c,int n,int m)
{
    int i;
    for(lbuc[0]=sbuc[0]=c[0],i=1;i<=m;i++)
    {
        lbuc[i]=c[i-1]+1;
        sbuc[i]=c[i];
    }
    for(i=1;i<=n;i++)
        if(sa[i]>1 && typ[sa[i]-1])
            sa[lbuc[s[sa[i]-1]]++]=sa[i]-1;
    for(i=n;i>=1;i--)
        if(sa[i]>1 && !typ[sa[i]-1])
            sa[sbuc[s[sa[i]-1]]--]=sa[i]-1;
}

```

```

void build_sa(int *s,int *sa,int *typ,int *c,int *p,int n,int m)
    // the last character of s[] must be 0
{
    int i;
    for(i=0;i<=m;i++) c[i]=0;
    for(i=1;i<=n;i++) c[s[i]]++;
    for(i=1;i<=m;i++) c[i]+=c[i-1];
    typ[n]=0;
    for(i=n-1;i>=1;i--)
        if(s[i]<s[i+1]) typ[i]=0;
        else if(s[i]>s[i+1]) typ[i]=1;
        else typ[i]=typ[i+1];
    int cnt=0;
    for(i=1;i<=n;i++)
        if(!typ[i] && (i==1 || typ[i-1])) p[++cnt]=i;
    for(i=1;i<=n;i++) sa[i]=0;
    for(i=0;i<=m;i++) sbuc[i]=c[i];
    for(i=1;i<=cnt;i++)
        sa[sbuc[s[p[i]]]--]=p[i];
    isort(s,sa,typ,c,n,m);
    int last=0,t=-1,x;
    for(i=1;i<=n;i++)
    {
        x=sa[i];
        if(!typ[x] && (x==1 || typ[x-1]))
        {
            if(!last || cmp(s,typ,x,last))
                name[x]=++t;
            else name[x]=t;
            last=x;
        }
    }
    for(i=1;i<=cnt;i++)
        s[n+i]=name[p[i]];
    if(t<cnt-1) build_sa(s+n,sa+n,typ+n,c+m+1,p+cnt,cnt,t);
    else
        for(i=1;i<=cnt;i++)
            sa[n+s[n+i]+1]=i;
    for(i=0;i<=m;i++) sbuc[i]=c[i];
    for(i=1;i<=n;i++) sa[i]=0;
    for(i=cnt;i>=1;i--)
        sa[sbuc[s[p[sa[n+i]]]--]=p[sa[n+i]];
    isort(s,sa,typ,c,n,m);
}

void build_hei()
{
    LL h=0,i,j;
    for(i=1;i<=n;i++)
        rk[sa[i]]=i;
    for(i=1;i<=n;i++)
    {
        if(rk[i]==n)
        {
            hei[n]=0;
            continue;

```

```

    }
    j=sa[rk[i]+1];
    if(h) h--;
    for(;j+h<=n && i+h<=n;h++)
        if(a[j+h]!=a[i+h]) break;
    hei[rk[i]]=h;
}
}

int main()
{
    scanf("%s",str);
    n=strlen(str);
    int i;
    for(i=1;i<=n;i++)
        a[i]=str[i-1];
    a[++n]=0; // the last character of a[] must be
               0
    build_sa(a,sa,typ,c,p,n,200);
    for(i=2;i<=n;i++)
        printf("%d%s",sa[i],i<n ? " " : "\n");
    return 0;
}

/* Count the number of distinct substrings:

#include <stdio>
#include <string>
#include <algorithm>

typedef long long LL;
const int LENGTH=100050,ALPHABET=100050;
char str[LENGTH];
int n,a[LENGTH*2],sa[LENGTH*2],typ[LENGTH*2],c[LENGTH+ALPHABET],
    p[LENGTH],sbuc[LENGTH+ALPHABET],lbuc[LENGTH+ALPHABET],name[
    LENGTH],hei[LENGTH],rk[LENGTH];
int l[100010],r[100010],stk[100010],last[100010];

inline int islms(int *typ,int i)
{
    return !typ[i] && (i==1 || typ[i-1]);
}

int cmp(int *s,int *typ,int p,int q)
{
    do
    {
        if(s[p]!=s[q]) return 1;
        p++; q++;
    }
    while(!islms(typ,p) && !islms(typ,q));
    return (!islms(typ,p) || !islms(typ,q) || s[p]!=s[q]);
}

void isort(int *s,int *sa,int *typ,int *c,int n,int m)
{

```



```

    int i;
    for(lbuc[0]=sbuc[0]=c[0], i=1; i<=m; i++)
    {
        lbuc[i]=c[i-1]+1;
        sbuc[i]=c[i];
    }
    for(i=1; i<=n; i++)
        if(sa[i]>1 && typ[sa[i]-1])
            sa[lbuc[s[sa[i]-1]]++] = sa[i]-1;
    for(i=n; i>=1; i--)
        if(sa[i]>1 && !typ[sa[i]-1])
            sa[sbuc[s[sa[i]-1]]--] = sa[i]-1;
}

void build_sa(int *s, int *sa, int *typ, int *c, int *p, int n, int m)
    // the last character of s[] must be 0
{
    int i;
    for(i=0; i<=m; i++) c[i]=0;
    for(i=1; i<=n; i++) c[s[i]]++;
    for(i=1; i<=m; i++) c[i]+=c[i-1];
    typ[n]=0;
    for(i=n-1; i>=1; i--)
        if(s[i]<s[i+1]) typ[i]=0;
        else if(s[i]>s[i+1]) typ[i]=1;
        else typ[i]=typ[i+1];
    int cnt=0;
    for(i=1; i<=n; i++)
        if(!typ[i] && (i==1 || typ[i-1])) p[++cnt]=i;
    for(i=1; i<=n; i++) sa[i]=0;
    for(i=0; i<=m; i++) sbuc[i]=c[i];
    for(i=1; i<=cnt; i++)
        sa[sbuc[s[p[i]]]--]=p[i];
    isort(s, sa, typ, c, n, m);
    int last=0, t=-1, x;
    for(i=1; i<=n; i++)
    {
        x=sa[i];
        if(!typ[x] && (x==1 || typ[x-1]))
        {
            if(!last || cmp(s, typ, x, last))
                name[x]=++t;
            else name[x]=t;
            last=x;
        }
    }
    for(i=1; i<=cnt; i++)
        s[n+i]=name[p[i]];
    if(t<cnt-1) build_sa(s+n, sa+n, typ+n, c+m+1, p+cnt, cnt, t);
    else
        for(i=1; i<=cnt; i++)
            sa[n+s[n+i]+1]=i;
    for(i=0; i<=m; i++) sbuc[i]=c[i];
    for(i=1; i<=n; i++) sa[i]=0;
    for(i=cnt; i>=1; i--)
        sa[sbuc[s[p[sa[n+i]]]]--]=p[sa[n+i]];
}

```

```

        isort(s,sa,typ,c,n,m);
    }

void build_hei()
{
    LL h=0,i,j;
    for(i=1;i<=n;i++)
        rk[sa[i]]=i;
    for(i=1;i<=n;i++)
    {
        if(rk[i]==n)
        {
            hei[n]=0;
            continue;
        }
        j=sa[rk[i]+1];
        if(h) h--;
        for(;j+h<=n && i+h<=n;h++)
            if(a[j+h]!=a[i+h]) break;
        hei[rk[i]]=h;
    }
}

int main()
{
    scanf("%d%s",&n,str);
    int i,j;
    for(i=1;i<=n;i++) a[i]=str[i-1];
    a[++n]=0;
    build_sa(a,sa,typ,c,p,n,200);
    build_hei();
    for(j=0,i=1;i<=n;i++)
    {
        while(j && hei[i]<=hei[stk[j]]) j--;
        l[i]=j ? stk[j] : 0;
        stk[++j]=i;
    }
    for(j=0,i=n;i>=1;i--)
    {
        while(j && hei[i]<=hei[stk[j]]) j--;
        r[i]=j ? stk[j] : n+1;
        stk[++j]=i;
    }
    memset(last,-1,sizeof(last));
    LL ans=0;
    for(hei[0]=hei[n+1]=0,i=1;i<=n;i++)
    {
        ans+=n-sa[i]-std::max(hei[i-1],hei[i]);
        if(l[i]>last[hei[i]])
            ans+=hei[i]-std::max(hei[l[i]],hei[r[i]]);
        // cnt[i]=r[i]-l[i];
        last[hei[i]]=i;
    }
    printf("%lld\n",ans);
    return 0;
}

```

```
*/
```

4.5 Manacher

```
void manacher()
{
    LL i,j,k;
    for(i=2,j=1,r[1]=0;i<=len;i++)
    {
        if(i<=j+r[j]) r[i]=min(r[j*2-i],j+r[j]-i);
        else r[i]=0;
        if(i+r[i]<j+r[j]) continue;
        for(k=r[i]+1;i+k<=len && i-k>=1;k++)
            if(str[i+k]==str[i-k]) r[i]++;
            else break;
        j=i;
    }
}
```

4.6 Palindromic automaton

```
int sz,son[N_NODE][ALPHABET],len[N_NODE],fail[N_NODE],cnt[N_NODE],pos[N_NODE];

void addnode(int l)
{
    sz++;
    memset(son[sz],0,sizeof(son[sz]));
    len[sz]=1; cnt[sz]=0;
}

void build(const string &str)
{
    sz=0;
    addnode(-1); addnode(0);
    fail[sz-1]=fail[sz]=sz-1;
    int i,u,x,l=str.size();
    for(u=1,i=1;i<=l;i++)
    {
        while(i-len[u]-1<1 || str[i-len[u]-1-1]!=str[i-1]) u=fail[u];
        x=str[i-1]-'a';
        if(!son[u][x])
        {
            addnode(len[u]+2);
            int v=fail[u];
            while(i-len[v]-1<1 || str[i-len[v]-1-1]!=str[i-1]) v=fail[v];
            fail[sz]=(son[v][x] ? son[v][x] : 2);
            son[u][x]=sz;
        }
    }
}
```

```

        cnt[son[u][x]]++;
        u=son[u][x];
    }
    for(i=sz;i>3;i--)
        cnt[fail[i]]+=cnt[i];
}

void match(const string &str)
{
    int i,u,x,l=str.size();
    for(u=1,i=1;i<=l;i++)
    {
        x=str[i-1]-'a';
        while(i-len[u]-1<1 || str[i-len[u]-1-1]!=str[i-1] || u>1
            && !son[u][x]) u=fail[u];
        u=son[u][x] ? son[u][x] : 1;
        pos[i]=u;
    }
}

```

5 Geometry

5.1 2-D vector

```

inline int dcmp(double x,double y)
{
    if(x>y+EPS) return 1;
    else if(x>y-EPS) return 0;
    else return -1;
}

struct V
{
    double x,y;
    // the operators are sorted by their priorities
    double operator*(const V &rhs) const // dot product
    {
        return x*rhs.x+y*rhs.y;
    }
    V operator/(double k) const // scalar division
    {
        return V{x/k,y/k};
    }
    V operator+(const V &rhs) const
    {
        return V{x+rhs.x,y+rhs.y};
    }
    V operator-(const V &rhs) const
    {
        return V{x-rhs.x,y-rhs.y};
    }
    bool operator<(const V &rhs) const // sort vectors by
        polar angle ([0, 2pi))

```

```

{
    return arg()<rhs.arg();
}
V operator&(double k) const    // scalar multiplication
{
    return V{x*k,y*k};
}
double operator^(const V &rhs) const    // cross product
{
    return x*rhs.y-y*rhs.x;
}
bool operator||(const V &rhs) const    // in the same
    direction
{
    return !dcmp(*this^rhs,0) && dcmp(*this*rhs,0)>0;
}
double len() const
{
    return sqrt(x*x+y*y);
}
double arg() const    // polar angle ([0, 2pi))
{
    double t=atan2(y,x);
    return dcmp(t,0)<0 ? t+PI*2 : t;
}
V rot(double alp) const    // rotate counterclockwise
{
    return V{x*cos(alp)-y*sin(alp),x*sin(alp)+y*cos(alp)};
}
};

```

5.2 2-D line

```

inline int dcmp(double x,double y)
{
    if(x>y+EPS) return 1;
    else if(x>y-EPS) return 0;
    else return -1;
}

struct LN
{
    V p,q;
    operator V() const
    {
        return q-p;
    }
    bool operator<(const LN &rhs) const    // sort lines by
        polar angle ([0, 2pi))
    {
        if((q-p)||rhs)
            return dcmp((rhs.p-p)^(rhs.q-p),0)>0;
        else return (q-p)<rhs;
    }
}

```

```

V inters(const LN &rhs) const
{
    if(!dcmp((q-p)^rhs,0)) return V{INF,INF};
    V t1=*this,t2=rhs,t3=rhs.p-p;
    return p+(t1&((t3^t2)/(t1^t2)));
}
LN shift(double l) const
{
    V d=q-p;
    d=d/d.len();
    V n=d.rot(PI/2);
    return LN{p+(n&l),q+(n&l)};
}
};

```

5.3 Area of polygon

```

// Include struct V (2D vector) and struct LN (2D line).

double area(V *arr,int n) // array of vertex
{
    int i;
    double res=0;
    for(i=2;i<n;i++)
        res+=(arr[i]-arr[1])^(arr[i+1]-arr[1])/2;
    return fabs(res);
}

V tmp[N_VERTEX];
double area(LN *ln,int n) // array of lines
{
    int i;
    for(i=1;i<=n;i++)
        tmp[i]=ln[i].inters(ln[i%n+1]);
    double res=0;
    for(i=2;i<n;i++)
        res+=(tmp[i]-tmp[1])^(tmp[i+1]-tmp[1])/2;
    return fabs(res);
}

```

5.4 Point-polygon distance

```

double distance(V o,LN *arr,int n)
{
    double res=INF;
    int i;
    for(i=1;i<=n;i++)
        if(dcmp(((arr[i].p-o)*arr[i])*((arr[i].q-o)*arr[i]),0)
            >0)
            res=std::min(res,std::min((arr[i].p-o).len(),(arr[i]
                ].q-o).len()));
}

```

```

        else res=std::min(res,fabs((arr[i].p-o)^(arr[i].q-o))/V(
            arr[i]).len());
    return res;
}

```

5.5 Segment+circle intersection

```

// Include struct V (2D vector).

int inters(V p,V q,V o,double r,V *res)
{
    double dx=q.x-p.x,dy=q.y-p.y;
    double a=sqr(dx)+sqr(dy),b=2*dx*(p.x-o.x)+2*dy*(p.y-o.y),c=
        sqr(p.x-o.x)+sqr(p.y-o.y)-sqr(r);
    double delta=sqr(b)-4*a*c;
    int cnt=0;
    if(delta>EPS)
    {
        double t1=(-b+sqr(delta))/a/2,t2=(-b-sqr(delta))/a/2;
        if(t1>-EPS && t1<1+EPS)      ////
            res[++cnt]=V{p.x+t1*dx,p.y+t1*dy};
        if(t2>-EPS && t2<1+EPS)      ////
            res[++cnt]=V{p.x+t2*dx,p.y+t2*dy};
        return cnt;
    }
    else if(delta>-EPS)      // delta=0
    {
        double t=-b/a/2;
        if(t>-EPS && t<1+EPS)      // To find line+circle
            intersections, delete the 3 'if's.
            res[++cnt]=V{p.x+t*dx,p.y+t*dy};
        return cnt;
    }
    else return 0;
}

```

5.6 Half-plane intersection

```

// Include struct V (2D vector) and struct LN (2D line).

inline int dcmp(double x,double y)
{
    if(x>y+EPS) return 1;
    else if(x>y-EPS) return 0;
    else return -1;
}

inline bool check(const LN &last,const LN &last2,const LN &l)
{
    V crs=last.inters(last2);
    return dcmp((l.p-crs)^(l.q-crs),0)>0;
}

```

```

}

int hp_inters(LN *arr,int n, LN *res)    // the intersection must
    be a polygon!
{
    std::sort(arr+1,arr+n+1);
    int i,front=1,rear=0;
    for(i=1;i<=n;i++)
    {
        if(front<=rear && (V(h[rear])||arr[i])) continue;
        while(front<rear && !check(h[rear],h[rear-1],arr[i]))
            rear--;
        while(front<rear && !check(h[front],h[front+1],arr[i]))
            front++;
        h[++rear]=arr[i];
    }
    while(front<rear && !check(h[rear],h[rear-1],h[front])) rear
        --;
    int m=0;
    for(i=front;i<=rear;i++) res[++m]=h[i];
    return m;
}

```

5.7 Convex hull

```

#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>

using namespace std;
const double PI=acos(-1);
struct VERTEX
{
    int x,y;
    VERTEX(){}
    VERTEX(int xx,int yy)
    {
        x=xx,y=yy;
    }
    bool operator< (const VERTEX &rhs) const
    {
        if(x!=rhs.x) return x<rhs.x;
        else return y<rhs.y;
    }
    VERTEX operator-(const VERTEX &rhs) const
    {
        return VERTEX(x-rhs.x,y-rhs.y);
    }
}
v[1010];
int n,h[2010];
inline double dist(const VERTEX &v1,const VERTEX &v2)
{

```



```

        return sqrt((v1.x-v2.x)*(v1.x-v2.x)+(v1.y-v2.y)*(v1.y-v2.y))
        ;
    }
    inline int cross(const VERTEX &v1,const VERTEX &v2)
    {
        return v1.x*v2.y-v1.y*v2.x;
    }

    int main()
    {
        scanf("%d",&n);
        int i;
        for(i=1;i<=n;i++)
            scanf("%d%d",&v[i].x,&v[i].y);
        sort(v+1,v+n+1);
        int sz=0,p=0;
        for(i=1;i<=n;i++)
        {
            while(p>=sz+2 && cross(v[h[p]]-v[h[p-1]],v[i]-v[h[p]])
                <0) p--;
            h[++p]=i;
        }
        sz=-p;
        for(i=n;i>=1;i--)
        {
            while(p>=sz+2 && cross(v[h[p]]-v[h[p-1]],v[i]-v[h[p]])
                <0) p--;
            h[++p]=i;
        }
        sz=-p;
        double ans=0;
        for(i=1;i<sz;i++)
            ans+=dist(v[h[i]],v[h[i%sz+1]]); // ans is the
            circumference of the hull
        printf("%.0f\n",ans);
        return 0;
    }

```

5.8 Rotating calipers

```

int n;
V v[N_VERTEX*2];

inline double area(int i,int j,int k)
{
    return fabs(v[i].x*v[j].y+v[j].x*v[k].y+v[k].x*v[i].y-v[j].x
        *v[i].y-v[k].x*v[j].y-v[i].x*v[k].y);
}

double maxarea()
{
    int i,j,k;
    double ans=0;
    for(i=1;i<=n;i++) v[i+n]=v[i];
}

```

```

    for(i=1;i<=n;i++)
        for(k=i+2,j=i+1;j<i+n-1;j++)
        {
            k=max(k,j+1);
            while(k<i+n-1 && dcmp(area(i,j,k),area(i,j,k+1))<0)
                k++;
            ans=max(ans,area(i,j,k));
        }
    return ans;
}

```

6 IO

6.1 Fast read

```

const int BUF_SZ=100000;
char buf[BUF_SZ+10];
inline char nc(void)
{
    static char *pr=buf,*pend=buf;
    if(pr==pend)
    {
        pr=buf;
        pend=pr+fread(buf,1,BUF_SZ,stdin);
        if(pr==pend) return EOF;
        else return *pr++;
    }
    return *pr++;
}
inline int readint(int *x)
{
    static char ch;
    ch=nc();
    while(ch!=EOF && (ch<'0' || ch>'9')) ch=nc();
    if(ch==EOF) return 0;
    for(*x=0;ch>='0' && ch<='9';ch=nc())
        *x=*x*10+ch-'0';
    return 1;
}

/*when reading from files, use "rb" mode!!!*/

```

6.2 Fast input (fin)

```

/*when reading from files, use "rb" mode!!!*/

class Fast_in
{
private:
    static const int BUF_SZ=100000;

```

```

char buf[BUF_SZ+10],*pr,*pend,ch;
bool state;
int len;
char nc()
{
    if(pr==pend)
    {
        pr=buf;
        pend=pr+fread(buf,1,BUF_SZ,stdin);
        if(pr==pend) return EOF;
        else return *pr++;
    }
    return *pr++;
}
bool isblank(char ch)
{
    if(ch==' ' || ch=='\t' || ch=='\n' || ch=='\r') return
        true;
    else return false;
}
bool isdigit(char ch)
{
    if(ch>='0' && ch<='9') return true;
    else return false;
}
public:
Fast_in()
{
    pr=pend=buf;
    state=true;
}
operator bool()
{
    return state;
}
Fast_in& get(char &ch)
{
    state=true;
    ch=nc();
    if(ch==EOF) state=false;
    return *this;
}

Fast_in& getline(char *s,int n,char delim='\n')
{
    state=true;
    for(len=0,ch=nc();len<n-1 && ch!=EOF && ch!=delim;len++,
        ch=nc()) *s++=ch;
    *s++='\0';
    if(ch==EOF) state=false;
    else if(len==n-1) *--pr=ch;
    return *this;
}

friend inline Fast_in& operator>>(Fast_in &fin,char *s)
{

```

```

static char ch;
fin.state=true;
do ch=fin.nc();
while(fin.isblank(ch));
if(ch==EOF)
{
    fin.state=false;
    return fin;
}
for(;ch!=EOF && !fin.isblank(ch);ch=fin.nc()) *s++=ch;
*s++='\0';
if(ch!=EOF) *--fin.pr=ch;
return fin;
}

template<class T>
friend inline Fast_in& operator>>(Fast_in &fin,T &x)
{
    static char ch;
    static int sig;
    fin.state=true;
    do ch=fin.nc();
    while(fin.isblank(ch));
    if(ch!='-' && !fin.isdigit(ch))
    {
        fin.state=false;
        if(ch!=EOF) *--fin.pr=ch;
        return fin;
    }
    if(ch=='-')
    {
        sig=-1;
        ch=fin.nc();
    }
    else sig=1;
    for(x=0;fin.isdigit(ch);ch=fin.nc())
        x=x*10+ch-'0';
    x*=sig;
    if(ch!=EOF) *--fin.pr=ch;
    return fin;
}
}fin;

```