# 1 ComputationalGeometry

## 1.1 operators

```
typedef std::pair<double,double> Pt;
#define X first
#define Y second
const double eps = 1e-6;
Pt point( double x , double y ){
    return make_pair( x , y );
}
Pt operator+( const Pt& p1 , const Pt& p2 ){
  return Pt( p1.X + p2.X , p1.Y + p2.Y );
}
Pt operator-( const Pt& p1 , const Pt& p2 ){
  return Pt( p1.X - p2.X , p1.Y - p2.Y );
}
double operator*( const Pt& p1 , const Pt& p2 ){
  return p1.X * p2.X + p1.Y * p2.Y;
}
double operator^( const Pt& p1 , const Pt& p2 ){
  return p1.X * p2.Y - p1.Y * p2.X;
}
Pt operator*( const Pt& p1 , const double& k ){
  return Pt( p1.X * k , p1.Y * k );
}
Pt operator/( const Pt& p1 , const double& k ){
  return Pt( p1.X / k , p1.Y / k );
}
bool equal( const double& a , const double& b ){
  return b - eps < a && a < b + eps;
}
bool less( const double& a , const double& b ){
  return a < b - eps;
}
bool lessOrEqual( const double& a , const double& b ){
  return a < b + eps;
}
double abs( const Pt& p1 ){
  return sqrt( p1 * p1 );
}
double area(){
  double sum = 0;
  for(int i = 0; i < n; i++) sum += 0.5*p[i]^[i+1];
  return sum;
}
Pt o;
D angle( const Pt& x ){
  return atan2( x.Y , x.X );
}
bool cmp_angle( Pt a , Pt b ){
  return angle( a - o ) < angle( b - o );
}
bool cmp_cross( Pt a , Pt b ){
  return ( a - o ) ^ ( b - o ) > 0;
}
```

## 1.2 inter_par

```
int ori( const Pt& o , const Pt& a , const Pt& b ){
  double cross = ( a - o ) ^ ( b - o );
  if( fabs( cross ) < eps ) return 0;
  return cross > 0 ? 1 : -1;
}
bool intersect(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){ //線段p1p2, p3p4
  return ori(p1, p2, p3)^ori(p1, p2, p4) < 0;
}
int parallel(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){
  return (p2-p1)^(p4-p3) == 0;
}
bool Collinear(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){ //共線
  ori(p1, p2, p3) == 0;
}
```

# 2 DivideConquer

## 2.1 PojTree

```
#include <bits/stdc++.h>

using namespace std;
typedef long long int ll;
typedef pair<int, ll> P;
#define idx first
#define w second

const int N = 10004;
const ll INF = (1ll << 60);

int vn;
ll k;
vector<P> graph[N];
vector<int> dist;
ll subtreeSz[N];
bool isCentroid[N];

void init()
{
    for(int i = 1; i <= vn; i++)
        graph[i].clear(), isCentroid[i] = false;
}

void buildTree()
{
    for(int i = 1; i < vn; i++)
    {
        int u, v, l; scanf("%d %d %d", &u, &v, &l);
        graph[u].push_back(P(v, l));
        graph[v].push_back(P(u, l));
    }
}

ll calSubsz(int v, int p)
{
    subtreeSz[v] = 1;
    for(auto c:graph[v])
    {
        if(isCentroid[c.idx] || c.idx == p) continue;
        subtreeSz[v] += calSubsz(c.idx, v);
    }
```

```
    return subtreeSz[v];
}


P getCentroid(int v, int p, ll subsz)
{
    P cen(-1, INF);
    ll mxsonSz = -1;
    for(auto c:graph[v])
    {
        if(c.idx == p || isCentroid[c.idx]) continue;
        P res = getCentroid(c.idx, v, subsz);
        if(res.w < cen.w) cen = res;
        mxsonSz = max(mxsonSz, subtreeSz[c.idx]);
    }
    mxsonSz = max(mxsonSz, subsz-subtreeSz[v]);
    if(mxsonSz < cen.w) cen = P(v, mxsonSz);
    return cen;
}

void getDist(int v, int p, ll w)
{
    if(w > k) return;
    dist.push_back(w);
    for(auto c:graph[v])
    {
        if(c.idx == p || isCentroid[c.idx]) continue;
        getDist(c.idx, v, w+c.w);
    }
}

ll calValidPair(int idx, ll w)
{
    dist.clear();
    getDist(idx, -1, w);
    sort(dist.begin(), dist.end());
    ll sum = 0;
    for(int l = 0, r = dist.size()-1; l < r; )
    {
        if(dist[r]+dist[l] <= k) sum += r-l, l++;
        else r--;
    }
    return sum;
}

ll treedc(int v)
{
    ll sum = 0;
    // find centroid
    calSubsz(v, v);
    int cen = getCentroid(v, v, subtreeSz[v]).idx;
    isCentroid[cen] = true;

    sum += calValidPair(cen, 0);
    for(auto c:graph[cen])
    {
        if(isCentroid[c.idx]) continue;
        sum -= calValidPair(c.idx, c.w);
        sum += treedc(c.idx);
    }
    return sum;
}

int main()
{
    while(scanf("%d %lld", &vn, &k) && vn && k)
    {
        init();
        buildTree();
        printf("%lld\n", treedc(1));
    }
    return 0;
}
```

## 2.2 nearestDist

```
bool cmp_y(P a, P b)
{
    return a.y < b.y;
}

bool cmp_x(P a, P b)
{
    return a.x < b.x;
}

double dc(P *arr, int n)
{
    if(n == 1) return INF;
    int mid = n/2;
    double cx = arr[mid].x;
    double dist = min( dc(arr, mid), dc(arr+mid, n-mid) );
    inplace_merge(arr, arr+mid, arr+n, cmp_y);
    static vector<P> brr; brr.clear();
    for(int i = 0; i < n; i++)
    {
        if(fabs(arr[i].x)-cx >= dist) continue;
        for(int j = brr.size()-1; j >= 0; j--)
        {
            double dx = brr[j].x-arr[i].x;
            double dy = brr[j].y-arr[i].y;
            if(fabs(dy) >= dist) break;
            dist = min(dist, sqrt(dx*dx+dy*dy));
        }
        brr.push_back(arr[i]);
    }
    return dist;
}

double nearestDist(P *arr, int n)
{
    sort(arr, arr+n, cmp_x);
    return dc(arr, n);
}
```

# 3 Flow

## 3.1 dinic

```
template<typename T>
struct DINIC{
  static const int MAXN=105;
  static const T INF=INT_MAX;
  int n, level[MAXN], cur[MAXN];
```

```
  struct edge{
    int v,pre;
    T cap,flow,r;
    edge(int v,int pre,T cap):v(v),pre(pre),cap(cap),flow(0),r(cap){}
  };
  int g[MAXN];
  vector<edge> e;
  void init(int _n){
    memset(g,-1,sizeof(int)*((n=_n)+1));
    e.clear();
  }
  void add_edge(int u,int v,T cap,bool directed=false){
    e.push_back(edge(v,g[u],cap));
    g[u]=e.size()-1;
    e.push_back(edge(u,g[v],directed?0:cap));
    g[v]=e.size()-1;
  }
  int bfs(int s,int t){
    memset(level,0,sizeof(int)*(n+1));
    memcpy(cur,g,sizeof(int)*(n+1));
    queue<int> q;
    q.push(s);
    level[s]=1;
    while(q.size()){
      int u=q.front();q.pop();
      for(int i=g[u];~i;i=e[i].pre){
        if(!level[e[i].v]&&e[i].r){
          level[e[i].v]=level[u]+1;
          q.push(e[i].v);
          if(e[i].v==t)return 1;
        }
      }
    }
    return 0;
  }
  T dfs(int u,int t,T cur_flow=INF){
    if(u==t)return cur_flow;
    T df;
    for(int &i=cur[u];~i;i=e[i].pre){
      if(level[e[i].v]==level[u]+1&&e[i].r){
        if(df=dfs(e[i].v,t,min(cur_flow,e[i].r))){
          e[i].flow+=df;
          e[i^1].flow-=df;
          e[i].r-=df;
          e[i^1].r+=df;
          return df;
        }
      }
    }
    return level[u]=0;
  }
  T dinic(int s,int t,bool clean=true){
    if(clean){
      for(size_t i=0;i<e.size();++i){
        e[i].flow=0;
        e[i].r=e[i].cap;
      }
    }
    T ans=0, mf=0;
    while(bfs(s,t))while(mf=dfs(s,t))ans+=mf;
    return ans;
  }
};
```

## 3.2 minCostMaxFlow

```
template<typename _T>
struct MCMF{
  static const int MAXN=440;
  static const _T INF=999999999;
  struct edge{
    int v,pre;
    _T cap,cost;
    edge(int v,int pre,_T cap,_T cost):v(v),pre(pre),cap(cap),cost(cost){}
  };
  int n,S,T;
  _T dis[MAXN],piS,ans;
  bool vis[MAXN];
  vector<edge> e;
  int g[MAXN];
  void init(int _n){
    memset(g,-1,sizeof(int)*((n=_n)+1));
    e.clear();
  }
  void add_edge(int u,int v,_T cap,_T cost,bool directed=false){
    e.push_back(edge(v,g[u],cap,cost));
    g[u]=e.size()-1;
    e.push_back(edge(u,g[v],directed?0:cap,-cost));
    g[v]=e.size()-1;
  }
  _T augment(int u,_T cur_flow){
    if(u==T||!cur_flow)return ans+=piS*cur_flow,cur_flow;
    vis[u]=1;
    _T r=cur_flow,d;
    for(int i=g[u];~i;i=e[i].pre){
      if(e[i].cap&&!e[i].cost&&!vis[e[i].v]){
        d=augment(e[i].v,min(r,e[i].cap));
        e[i].cap-=d;
        e[i^1].cap+=d;
        if(!(r-=d))break;
      }
    }
    return cur_flow-r;
  }
  bool modlabel(){
    for(int u=0;u<=n;++u)dis[u]=INF;
    static deque<int>q;
    dis[T]=0,q.push_back(T);
    while(q.size()){
      int u=q.front();q.pop_front();
      _T dt;
      for(int i=g[u];~i;i=e[i].pre){
        if(e[i^1].cap&&(dt=dis[u]-e[i].cost)<dis[e[i].v]){
          if((dis[e[i].v]=dt)<=dis[q.size()?q.front():S]){
            q.push_front(e[i].v);
          }else q.push_back(e[i].v);
        }
      }
    }
    for(int u=0;u<=n;++u)
      for(int i=g[u];~i;i=e[i].pre)
        e[i].cost+=dis[e[i].v]-dis[u];
    return piS+=dis[S], dis[S]<INF;
  }
  _T mincost(int s,int t){
    S=s,T=t;
    piS=ans=0;
    while(modlabel()){
      do memset(vis,0,sizeof(bool)*(n+1));
```

```
      while(augment(S,INF));
    }return ans;
  }
};
```

# 4 Graph

## 4.1 floyd_warshall

```
int d[N][N];

void init()
{
    for(int i = 0; i < v; i++)
        for(int j = 0; j < v; j++)
            if(i == j) d[i][j] = 0;
            else d[i][j] = INF;
}

void floyd_warshall()
{
    for(int k = 0; k < v; k++)
        for(int i = 0; i < v; i++)
            for(int j = 0; j < v; j++)
                if(d[i][k] != INF && d[k][j] != INF)
                    d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}
```

## 4.2 mst_prim

```
int cost[100][100];
bool used[100];
int mincost[100];
int v, e;
#define INF 2147483647

int prim()
{
    for(int i = 0; i < v; i++)
    {
        mincost[i] = INF;
        used[i] = false;
    }
    mincost[0] = 0;
    int res = 0;

    while(true)
    {
        int x = -1;
        for(int u = 0; u < v; u++)
            if(!used[u] && (x == -1 || mincost[u] < mincost[x])) x = u;

        if(x == -1) break;
        used[x] = true;
        res += mincost[x];

        for(int u = 0; u < v; u++)
            mincost[u] = min(mincost[u], cost[x][u]);
    }
    return res;
}

void init()
{
    for(int i = 0; i < v; i++)
        for(int j = 0; j < v; j++)
            if(i == j) cost[i][j] = 0;
            else cost[i][j] = INF;
}
```

## 4.3 mst_kruskal

```
struct edge { int u, v, cost; };

bool comp(const edge& e1, const edge& e2)
{
    return e1.cost < e2.cost;
}

int kruskal()
{
    sort(es, es + e, comp);
    dset s(v);
    int res = 0;
    for(int i = 0; i < e; i++)
    {
        edge E = es[i];
        if(s.Find(E.u) != s.Find(E.v))
        {
            s.Union(E.u, E.v);
            res += E.cost;
        }
    }
    return res;
}
```

## 4.4 bellman_Ford

```
struct edge{ int from, to, cost; };
#define INF 2147483647

edge es[100];

int d[100]; //min distance
int V, E, s, f;

bool bellman_ford()  // return true if there is negative loop
{
    for(int i = 0; i < V; i++) d[i] = INF;
    d[s] = 0;

    for(int i = 0; i < V; i++)
    {
        for(int j = 0; j < E; j++)
        {
            edge e = es[j];
            if(d[e.from] != INF && d[e.to] > d[e.from] + e.cost)
            {
```

```
                d[e.to] = d[e.from] + e.cost;
                if(i == V - 1) return true; //got neg loop
            }
            if(d[e.to] != INF && d[e.from] > d[e.to] + e.cost)
            {
                d[e.from] = d[e.to] + e.cost;
                if(i == V - 1) return true; //got neg loop
            }
        }
    }
    return false;
}
```

## 4.5 dijkstra

```

struct edge{int to, cost;};
typedef pair<int, int> P; //first = min distance, second = v id
#define f first
#define s second
#define INF 2147483647

int V, E, S, F;
vector<edge> G[100];
int d[100];

void dijkstra()
{
    priority_queue<P, vector<P>, greater<P>> q;
    fill(d, d + V, INF);
    d[S] = 0;
    q.push(P(0, S));

    while(!q.empty())
    {
        P p = q.top(); q.pop();
        int v = p.s;
        if(d[v] < p.f) continue;
        for(int i = 0; i < G[v].size(); i++)
        {
            edge e = G[v][i];
            if(d[e.to] > d[v] + e.cost)
            {
                d[e.to] = d[v] + e.cost;
                q.push(P(d[e.to], e.to));
            }
        }
    }
}
```

## 4.6 spfa

```
typedef pair<int, ll> P;
#define idx first
#define w second
int vn, en;
vector<P> graph[N];
ll dist[N];

bool spfa()  // return true if neg cycle
{
    for(int i = 0; i < vn; i++) dist[i] = INF; dist[0] = 0;
    int cnt[N] = {0};
    bool inq[N] = {false};
    queue<int> q; q.push(0); inq[0] = true;
    while(!q.empty())
    {
        int s = q.front(); q.pop();
        inq[s] = false;
        for(auto e:graph[s])
        {
            if(dist[e.idx] > dist[s]+e.w)
            {
                dist[e.idx] = dist[s]+e.w;
                if(++cnt[e.idx] >= vn) return true;
                if(!inq[e.idx])
                {
                    inq[e.idx] = true;
                    q.push(e.idx);
                }
            }
        }
    }
    return false;
}
```

# 5 Math

## 5.1 exgcd

```
ll exgcd(ll a, ll &ar, ll b)  //維護a*ar+b*as=gcd(a, b)
{
   ll as = 0, br = 0, bs = 1;
   while(a && b)
   {
      ar -= br*(a/b);
      as -= bs*(a/b);
      a %= b;
      if(a == 0) break;
      br -= ar*(b/a);
      bs -= as*(b/a);
      b %= a;
   }
   if(a == 0) a = b, ar = br;  //維護a*ar+b*as=gcd(a, b)
   return a; //return gcd(a, b)
}
```

## 5.2 NTT

```
typedef long long ll;

const ll P = (479<<21)+1;
const ll G = 3;
inline ll fpw(ll x, ll y, ll m)
{
  ll rtn = 1;
  for(x=(x>=m?x%m:x);y;y>>=1)
```

```
  {
    if(y&1) rtn = rtn*x%m;
    x = x*x%m;
  }
  return rtn;
}
inline vector<ll> ntt(vector<ll> rtn, int Rev = 1)
{
  int ntt_n = rtn.size();
  for(int i=0,j=0;i<ntt_n;i++)
  {
    if(i>j) swap(rtn[i],rtn[j]);
    for(int k=(ntt_n>>1);(j^=k)<k;k>>=1);
  }
  for(int i=2,m=1;i<=ntt_n;i<<=1,m++)
  {
    ll w = 1, wn = fpw(G,(P-1)>>m,P), u, t;
    int mh = i>>1;
    for(int j=0;j<mh;j++)
    {
      for(int k=j;k<ntt_n;k+=i)
      {
        u = rtn[k], t = w*rtn[k+mh]%P;
        rtn[k] = (u+t)%P;
        rtn[k+mh] = (u-t+P)%P;
      }
      w = w*wn%P;
    }
  }
  if(!~Rev)
  {
    for(int i=1;i<ntt_n/2;i++) swap(rtn[i],rtn[ntt_n-i]);
    ll Revn = fpw(ntt_n,P-2,P);
    for(int i=0;i<ntt_n;i++) rtn[i] = rtn[i]*Revn%P;
  }
  return rtn;
}
```

## 5.3 BigInteger

```
class BigInt
{
  public:
    // constructors
    BigInt();
    BigInt(ll);
    BigInt(const char*);
    BigInt(string);
    BigInt(bool, vector<int>);
    // functions
    inline void print();
    // operators
    bool operator== (const BigInt &a) const;
    bool operator!= (const BigInt &a) const;
    bool operator< (const BigInt &a) const;
    bool operator> (const BigInt &a) const;
    bool operator<= (const BigInt &a) const;
    bool operator>= (const BigInt &a) const;
    BigInt operator- () const;
    BigInt operator+ (const BigInt &a) const;
    BigInt operator- (const BigInt &a) const;
    BigInt operator* (const BigInt &a) const;
    BigInt operator/ (const BigInt &a) const;
    BigInt operator% (const BigInt &a) const;
    // variables
    const static int MAX = 1000000;
    bool Neg = false;
    vector<int> seq;
};

// constructors
BigInt::BigInt(){}
BigInt::BigInt(ll in)
{
  if(in<0) Neg=true, in=-in;
  while(in!=0)
  {
    seq.emplace_back(in%MAX);
    in/=MAX;
  }
  if(seq.empty()) seq.emplace_back(0);
}
BigInt::BigInt(const char *s)
{
  int i, j, tmp, end=0;
  if(s[0]=='-') Neg=true, end=1;
  for(i=strlen(s)-1, j=1, tmp=0; i>=end; i--, j*=10)
  {
    if(j==MAX)
    {
      seq.emplace_back(tmp);
      j=1, tmp=0;
    }
    tmp += (s[i]-'0')*j;
  } seq.emplace_back(tmp);
}
BigInt::BigInt(string s):BigInt(s.c_str()){}
BigInt::BigInt(bool b, vector<int> v):Neg(b),seq(v){}
// functions
void BigInt::print()
{
  if(Neg) putchar('-');
  printf("%d",seq.back());
  for(int i=(int)(seq.size())-2; i>=0; i--)
    printf("%06d",seq[i]);
  puts("");
}
// operators
bool BigInt::operator== (const BigInt &a) const
{
  return Neg==a.Neg&&seq==a.seq;
}
bool BigInt::operator!= (const BigInt &a) const
{
  return !((*this)==a);
}
bool BigInt::operator< (const BigInt &a) const
{
  if(Neg^a.Neg) return Neg;
  if(seq.size()!=a.seq.size()) return Neg^(seq.size()<a.seq.size());
  for(int i=seq.size()-1; i>=0; i--)
    if(seq[i]!=a.seq[i]) return Neg^(seq[i]<a.seq[i]);
  return false;
}
bool BigInt::operator> (const BigInt &a) const
{
  if(Neg^a.Neg) return a.Neg;
  if(seq.size()!=a.seq.size()) return a.Neg^(seq.size()>a.seq.size());
  for(int i=seq.size()-1; i>=0; i--)
```

```
    if(seq[i]!=a.seq[i]) return a.Neg^(seq[i]>a.seq[i]);
  return false;
}
bool BigInt::operator<= (const BigInt &a) const
{
  return !((*this)>a);
}
bool BigInt::operator>= (const BigInt &a) const
{
  return !((*this)<a);
}
BigInt BigInt::operator- () const
{
  return BigInt(Neg^1, seq);
}
BigInt BigInt::operator+ (const BigInt &a) const
{
  if(Neg^a.Neg)
    return Neg?a-(-(*this)):(*this)-(-a);
  BigInt rtn(Neg, vector<int>(max(seq.size(),a.seq.size())));
  for(int i=0; i<(int)(seq.size()); i++) rtn.seq[i]+=seq[i];
  for(int i=0; i<(int)(a.seq.size()); i++) rtn.seq[i]+=a.seq[i];
  for(int i=0; i<(int)(rtn.seq.size())-1; i++)
    if(rtn.seq[i]>=MAX)
      rtn.seq[i+1]+=rtn.seq[i]/MAX, rtn.seq[i]%=MAX;
  if(rtn.seq.back()>=MAX)
  {
    rtn.seq.emplace_back(rtn.seq.back()/MAX);
    rtn.seq[rtn.seq.size()-2]%=MAX;
  }
  return rtn;
}
BigInt BigInt::operator- (const BigInt &a) const
{
  if(Neg^a.Neg) return (*this)+(-a);
  if(Neg^((*this)<a)) return (-a)-(-(*this));
  BigInt rtn(Neg, vector<int>(max(seq.size(),a.seq.size())));
  for(int i=0; i<(int)(seq.size()); i++) rtn.seq[i]+=seq[i];
  for(int i=0; i<(int)(a.seq.size()); i++) rtn.seq[i]-=a.seq[i];
  for(int i=0; i<(int)(rtn.seq.size())-1; i++)
    if(rtn.seq[i]<0)
      rtn.seq[i+1]--, rtn.seq[i]+=MAX;
  while(!rtn.seq.empty()&&!rtn.seq.back()) rtn.seq.pop_back();
  if(rtn.seq.empty()) rtn = BigInt(0ll);
  return rtn;
}
BigInt BigInt::operator* (const BigInt &a) const
{
  BigInt rtn(Neg^a.Neg, vector<int>(0));
  vector<Complex> x, y;
  for(auto i:seq) x.emplace_back(i);
  for(auto i:a.seq) y.emplace_back(i);
  int N=1;
  while(N<(int)(x.size()+y.size())) N <<= 1;
  while(N!=(int)(x.size())) x.emplace_back(0);
  while(N!=(int)(y.size())) y.emplace_back(0);
  x = fft(x), y = fft(y);
  for(int i=0; i<N; i++) x[i] = x[i]*y[i];
  x = fft(x,-1);
  ll tmp = 0;
  for(int i=0; i<N; i++)
  {
    tmp += (ll)(x[i].x+0.1);
    rtn.seq.emplace_back(tmp%MAX);
    tmp /= MAX;
  } rtn.seq.emplace_back(tmp);
  while(!rtn.seq.empty()&&!rtn.seq.back()) rtn.seq.pop_back();
  if(rtn.seq.empty()) rtn = BigInt(0ll);
  return rtn;
}
BigInt BigInt::operator/ (const BigInt &a) const
{
  if(a==BigInt(0ll)) return a;
  BigInt rtn, check, BItmp, posiA, posiB;
  posiA = (*this), posiA.Neg = false;
  posiB = a, posiB.Neg = false;
  int PRECISION = max(seq.size(),a.seq.size())+6, N = 1;
  ll tmp = 0;
  while(N<PRECISION+6) N <<= 1;
  N <<= 1;
  vector<Complex> B, c1(N,0), c2(N,0), c3(N,0);
  vector<Complex> *x = (&c1), *xp = (&c2), *calc = (&c3);
  for(int i=a.seq.size()-1; i>=0; i--)
    B.emplace_back(a.seq[i]);
  B.resize(N,0);
  B = fft(B);
  (*x)[a.seq.size()] = MAX/a.seq.back();
  bool found = false;
  while(!found)
  {
    (*x) = fft(*x);
    for(int i=0; i<N; i++) (*calc)[i] = (*x)[i]*B[i];
    (*calc) = fft(*calc,-1);
    for(int i=a.seq.size()-1; i<N; i++)
      (*calc)[i-a.seq.size()+1] = (*calc)[i];
    for(int i=N-a.seq.size()+1; i<N; i++)
      (*calc)[i] = 0;
    for(int i=N-1; i>=1; i--)
    {
      tmp = (ll)((*calc)[i].x+0.1);
      (*calc)[i-1] = (ll)((*calc)[i-1].x+0.1)+tmp/MAX;
      (*calc)[i] = tmp %= MAX;
      if(tmp)
      {
        (*calc)[i-1] = (ll)((*calc)[i-1].x+0.1)+1;
        (*calc)[i] = MAX-tmp;
      }
    } (*calc)[0] = 2-(ll)((*calc)[0].x+0.1);
    for(int i=PRECISION+6; i<N; i++) (*calc)[i] = 0;
    (*calc) = fft(*calc);
    for(int i=0; i<N; i++) (*xp)[i] = (*calc)[i]*(*x)[i];
    (*xp) = fft(*xp,-1);
    (*x) = fft(*x,-1);
    for(int i=N-1; i>=1; i--)
    {
      tmp = (ll)((*xp)[i].x+0.1);
      (*xp)[i] = tmp%MAX;
      (*xp)[i-1] = (ll)((*xp)[i-1].x+0.1)+tmp/MAX;
    }
    for(int i=PRECISION+6; i<N; i++) (*xp)[i] = 0;
    found = true;
    for(int i=0; i<=PRECISION&&found; i++)
      if((ll)((*xp)[i].x+0.1)!=(ll)((*x)[i].x+0.1))
        found = false;
    calc = x, x = xp, xp = calc, calc = (&c3);
  }
  for(int i=N-1; i>=(int)(a.seq.size())-1; i--)
    rtn.seq.emplace_back((ll)((*x)[i].x+0.1));
  while(!rtn.seq.back()) rtn.seq.pop_back();
  rtn = rtn*posiA;
  for(int i=N-1; i<(int)(rtn.seq.size()); i++)
    rtn.seq[i-N+1] = rtn.seq[i];
```

```
  for(int i=max((int)(rtn.seq.size()-N+1),0); i<(int)(rtn.seq.size()); i++)
    rtn.seq[i] = 0;
  while(!rtn.seq.empty()&&!rtn.seq.back()) rtn.seq.pop_back();
  if(rtn.seq.empty()) rtn = BigInt(0ll);
  check = rtn*posiB;
  BItmp = check+posiB;
  while(posiA>=BItmp)
    rtn = rtn+BigInt(1), check = BItmp, BItmp = check+posiB;
  BItmp = check-posiB;
  while(posiA<=BItmp)
    rtn = rtn-BigInt(1), check = BItmp, BItmp = check-posiB;
  rtn.Neg = Neg^a.Neg;
  return rtn;
}
BigInt BigInt::operator% (const BigInt &a) const
{
  return (*this)-((*this)/a)*a;
}
```

## 5.4 prime_detect

```
const int N = 10000000;
bool isprime[N] = {true};
void prime_detect()
{
    for(int i = 2; i < sq; i++)
        if(isprime[i])
            for(int j = i*i; j < N; j+=i) isprime[j] = false;
}
```

## 5.5 modeq

```
// 解線性模方程組(最小非負整數解)
const int N;  //N個方程
ll A[N], B[N], M[N];  // A * X = B (%M)
ll solve() //解X, return INF if no solution
{
   ll k = 0, h = 1;
   for(ll i = 0; i < N; i++)
   {
      ll a = A[i]*h, b = B[i]-A[i]*k, m = M[i], ar;
      ll d = exgcd(a, ar=1, m);
      if(b%d != 0) return INF;
      ll n = abs(m/d);
      ll t = ar*b/d; t%=n; t+=n; t%=n;

      k += h*t, h *= n; k%=h; //維護解是正的
   }
   int ret = (k%h+h)%h;
   return ret;
}
```

## 5.6 FFT

```
const double PI = acos(-1.0);
struct Complex
{
  double x,y;
  Complex(){}
  Complex(double a):x(a),y(0){}
  Complex(double a, double b):x(a),y(b){}
  Complex operator+ (const Complex &a){ return Complex(x+a.x,y+a.y); }
  Complex operator- (const Complex &a){ return Complex(x-a.x,y-a.y); }
  Complex operator* (const Complex &a){ return Complex(x*a.x-y*a.y,x*a.y+y*a.x); }
};
inline vector<Complex> fft(vector<Complex> rtn, int Rev = 1)
{
  int fft_n = rtn.size();
  for(int i=0,j=0;i<fft_n;i++)
  {
    if(i>j) swap(rtn[i],rtn[j]);
    for(int k=(fft_n>>1);(j^=k)<k;k>>=1);
  }
  for(int i=2,m;i<=fft_n;i<<=1)
  {
    m = i>>1;
    for(int j=0;j<fft_n;j+=i)
    {
      for(int k=0;k<m;k++)
      {
        Complex y = rtn[j+k+m]*Complex(cos(2*PI/i*k), Rev*sin(2*PI/i*k));
        rtn[j+k+m] = rtn[j+k]-y;
        rtn[j+k] = rtn[j+k]+y;
      }
    }
  }
  for(int i=0;!~Rev&&i<fft_n;i++)
    rtn[i].x = rtn[i].x/fft_n;
  return rtn;
}
```

# 6 String

## 6.1 SuffixArray-STL

```
struct CMP
{
  int len,k,*Rank,a,b;
  inline bool operator()(int i, int j)
  {
    if(Rank[i]!=Rank[j])return Rank[i]<Rank[j];
    a=(i+=k)<len?Rank[i]:-1;
    b=(j+=k)<len?Rank[j]:-1;
    return a<b;
  }
};
void SA_build(int *SA, int *Rank, char *S){
  int tmp[MAX_N], len=strlen(S);
  for(int i=0;i<len;i++) SA[i]=i, Rank[i]=S[i];
  CMP cmp={len,1};
  while(cmp.k*=2)
  {
    cmp.Rank=Rank;
    sort(SA,SA+len,cmp);
    tmp[SA[0]]=0;
```

```
    for(int i=1;i<len;i++)
      tmp[SA[i]]=tmp[SA[i-1]]+cmp(SA[i-1],SA[i]);
    if(tmp[SA[len-1]]==len-1) break;
    for(int i=0;i<len;i++) Rank[i]=tmp[i];
  }
}
```

## 6.2 Z-value

```
void Z_build(const char *S, int *Z)
{
  Z[0]=0;
  int b=0;
  for(int i=1;S[i];i++)
  {
    if(Z[b]+b<i) Z[i]=0;
    else Z[i]=min(Z[b]+b-i,Z[i-b]);
    while(S[i+Z[i]]&&S[Z[i]]==S[i+Z[i]]) Z[i]++;
    if(Z[i]+i>Z[b]+b) b=i;
  }
}
```

## 6.3 LCP

```
//build query in O(nlogn), query LCP(i,j) in O(1)
int dp_height[MAX_N][20];
void height_build(int *SA, int *Rank, char *S, int *Height)
{
  int len=strlen(S), k=0;
  for(int i=0;i<len;i++)
  {
    if(Rank[i]==0) continue;
    while(S[i+k] == S[SA[Rank[i]-1]+k]) k++;
    Height[Rank[i]]=k;
    if(k) k--;
  } Height[0]=0;
  for(int i=0;i<len;i++) dp_height[i][0]=Height[i];
  for(int i=0;i<len;i++) for(int j=1;i+(1<<j)<len;j++)
    dp_height[i][j]=min(dp_height[i][j-1], dp_height[i+(1<<(j-1))][j-1]);
}
int height_query(int x, int y)
{
  int k=0;
  while((1<<(k+1))<=y-x) k++;
  return min(dp_height[x+1][k], dp_height[y-(1<<k)+1][k]);
}
```

## 6.4 SuffixArray

```
void SA_radix_sort(int *s, int *e, int *Rank, int rankcnt)
{
  int box[MAX_N], tmp[MAX_N], len=e-s;
  memset(box,0,sizeof(int)*rankcnt);
  for(int i=0;i<len;i++) box[Rank[i]]++;
  for(int i=1;i<rankcnt;i++) box[i]=box[i]+box[i-1];
  for(int i=len-1;i>=0;i--) tmp[--box[Rank[s[i]]]]=s[i];
  for(int i=0;i<len;i++) s[i]=tmp[i];
}
#define equal(a,b,c) c[a]!=c[b]||a+k>=len||c[a+k]!=c[b+k]
void SA_build(int *SA, int *Rank, char *S)
{
  int ranktmp[MAX_N], len=strlen(S), rankcnt='z'+1;
  for(int i=0;i<len;i++) Rank[i]=S[i];
  for(int k=1;rankcnt!=len;k*=2)
  {
    for(int i=0;i<len;i++) SA[i]=(i+len-k)%len;
    SA_radix_sort(SA+k, SA+len, Rank+k, rankcnt);
    SA_radix_sort(SA, SA+len, Rank, rankcnt);
    ranktmp[SA[0]]=0, rankcnt=0;
    for(int i=1;i<len;i++)
      ranktmp[SA[i]]=rankcnt+=equal(SA[i-1], SA[i], Rank);
    rankcnt++;
    for(int i=0;i<len;i++) Rank[i]=ranktmp[i];
  }
}
#undef equal
```

## 6.5 KMP

```
void failure_build(const char *p, int *fail)
{
  for(int i=1, j=fail[0]=-1; p[i]; i++)
  {
    while(j>=0&&p[j+1]!=p[i]) j=fail[j];
    if(p[j+1]==p[i]) j++;
    fail[i]=j;
  }
}
int KMP(const char *T, const char *P, int *fail)
{
  failure_build(P, fail);
  for(int i=0, j=-1; T[i]; i++)
  {
    while(j>=0&&P[j+1]!=T[i]) j=fail[j];
    if(P[j+1]==T[i]) j++;
    if(!P[j+1]) return i-j;
  }
  return -1;
}

//使用方法: KMP(主字串, 待匹配字串, failure array)
//回傳: 第一個完全匹配的位置
```

# 7 Tree

## 7.1 treap

```
struct Treap
{
    int pri, sz;
    int rev;
    ll data, sum;   // tag: make-same
    Treap *lchild, *rchild;
```

```
    Treap(ll d):pri(rand()), sz(1), rev(0), data(d), sum(d), lchild(NULL), rchild(NULL)
    {
    }
    inline void up();
    inline void down();
};

inline int size(Treap *t) { return t? t->sz:0; }
inline ll get_data(Treap *t) { return t? t->data:0; }
inline ll get_sum(Treap *t) { return t? t->sum:0; }

inline void Treap::up()
{
    if(lchild) lchild->down();
    if(rchild) rchild->down();
    sz = 1+size(lchild)+size(rchild);
    sum = get_sum(lchild) + data + get_sum(rchild);
}

inline void Treap::down()
{
    if(rev)
    {
        swap(mxpre, mxpost);
        swap(lchild, rchild);
        if(lchild) lchild->rev ^= 1;
        if(rchild) rchild->rev ^= 1;
        rev ^= 1;
    }
}

Treap *merge(Treap *a, Treap *b)
{
    if(!a || !b) return (a? a:b);
    if(a->pri < b->pri)
    {
        a->down();
        a->rchild = merge(a->rchild, b);
        a->up();
        return a;
    }
    else
    {
        b->down();
        b->lchild = merge(a, b->lchild);
        b->up();
        return b;
    }
}

void split(Treap *o, Treap *&a, Treap *&b, int k)
{
    if(!o) a = b = NULL;
    else
    {
        o->down();
        if(k >= size(o->lchild)+1)
        {
            a = o;
            split(o->rchild, a->rchild, b, k-size(o->lchild)-1);
        }
        else
        {
            b = o;
            split(o->lchild, a, b->lchild, k);
        }
        o->up();
    }
}
```

## 7.2 HeavyLightDecomposition

```
const int MAX_N;
vector<int> link[MAX_N]; //edge

void dfs_build(int now, int fa, int *weight, int *depth, int *pa, int *son)
{
  weight[now]=1;
  son[now]=-1;
  pa[now]=fa;
  for(auto i:link[now])
  {
    if(i==fa) continue;
    depth[i]=depth[now]+1;
    dfs_build(i,now,weight,depth,pa,son);
    if(son[now]==-1||weight[son[now]]<weight[i]) son[now]=i;
    weight[now]+=weight[i];
  }
}
void build_top(int now, int top,int *pa, int *son, int *link_top)
{
  link_top[now]=top;
  if(son[now]==-1) return;
  build_top(son[now],top,pa,son,link_top);
  for(auto i:link[now])
  {
    if(i==son[now]||i==pa[now]) continue;
    build_top(i,i,pa,son,link_top);
  }
}
inline void HLD(int *weight, int *depth, int *pa, int *son, int *link_top)
{
  memset(son,-1,sizeof(int)*MAX_N);
  depth[1]=1; //set node(1) as root
  dfs_build(1,0,weight,depth,pa,son);
  build_top(1,1,pa,son,link_top);
}
inline int find_lca(int x, int y, int *depth, int *pa, int *link_top)
{
  int tx=link_top[x], ty=link_top[y];
  while(tx!=ty)
  {
    if(depth[tx]<depth[ty])
    {
      swap(tx,ty);
      swap(x,y);
    }
    tx=link_top[x=pa[x]];
  }
  return depth[x]<depth[y]?x:y;
}
//usage:
//build HeavyLightDecomposition: HLD
//find LCA(x,y): find_lca
```

## 7.3 disjoint_set

```cpp
// path compression
int f[N];

int findrt(int x)
{
    if(f[x] == x) return x;
    else return f[x] = findrt(f[x]);
}

int same(int x, int y)
{
    return findrt(x) == findrt(y);
}

void uni(int x, int y)
{
    f[findrt(y)] = findrt(x);
}

void init()
{
  for(int i = 0; i < N; i++) f[i] = i;
}

//union by rank
int f[N];  //disjoint set
int rk[N]; //union by rank

int findrt(int x)
{
    if(f[x] == x) return x;
    else return f[x] = findrt(f[x]);
}

bool same(int x, int y)
{
    return findrt(x) == findrt(y);
}

void uni(int x, int y)
{
    x = findrt(x), y = findrt(y);
    if(x == y) return;
    if(rk[x] < rk[y]) f[x] = y;
    else if(rk[x] == rk[y]) f[x] = y, rk[y]++;
    else f[y] = x;
}

void init()
{
  for(int i = 0; i < N; i++) f[i] = i, rank[i] = 0;
}
```

## 7.4 2d_st_tag

```cpp
//二維陣列單點查詢區間加值
class St1d
{
private:
    ll st[4*N];

public:
    void build();
    void modify(int l, int r, int idx, int L, int R, ll v);
    ll query(int l, int r, int idx, int x);
    void down(int idx);
};

void St1d::build()
{
    memset(st, 0, sizeof(st));
}

void St1d::modify(int l, int r, int idx, int L, int R, ll v)
{
    if(r < L || R < l) return;
    if(L <= l && r <= R)
    {
        st[idx] += v;
        return;
    }
    assert(l != r);
    down(idx);
    int mid = (l+r)/2;
    modify(l, mid, idx*2, L, R, v);
    modify(mid+1, r, idx*2+1, L, R, v);
}

ll St1d::query(int l, int r, int idx, int x)
{
    if(x < l || r < x) return 0;
    if(l == x && r == x) return st[idx];
    down(idx);
    int mid = (l+r)/2;
    ll left = query(l, mid, idx*2, x);
    ll right = query(mid+1, r, idx*2+1, x);
    return left+right;
}

void St1d::down(int idx)
{
    st[idx*2] += st[idx], st[idx*2+1] += st[idx];
    st[idx] = 0;
}

//////////////////////

class St2d
{
private:
    St1d st[4*N];

public:
    void build(int il, int ir, int idx);
    void modify(int il, int ir, int jl, int jr, int idx, int iL, int iR, int jL, int jR, ll v
        );
    ll query(int il, int ir, int jl, int jr, int idx, int i, int j);
};

void St2d::build(int il, int ir, int idx)
{
    st[idx].build();
    if(il == ir) return;
    int mid = (il+ir)/2;
    build(il, mid, idx*2);
    build(mid+1, ir, idx*2+1);
}
```

```
void St2d::modify(int il, int ir, int jl, int jr, int idx, int iL, int iR, int jL, int jR, ll
      v)
{
    if(ir < iL || iR < il) return;
    if(iL <= il && ir <= iR)
    {
        st[idx].modify(jl, jr, 1, jL, jR, v); return;
    }
    int mid = (il+ir)/2;
    modify(il, mid, jl, jr, idx*2, iL, iR, jL, jR, v);
    modify(mid+1, ir, jl, jr, idx*2+1, iL, iR, jL, jR, v);
}

ll St2d::query(int il, int ir, int jl, int jr, int idx, int i, int j)
{
    ll tot = 0;
    if(i < il || ir < i) return 0;
    if(il <= i && i <= ir) tot += st[idx].query(jl, jr, 1, j);
    if(il == i && ir == i) return tot;
    int mid = (il+ir)/2;
    tot += query(il, mid, jl, jr, idx*2, i, j);
    tot += query(mid+1, ir, jl, jr, idx*2+1, i, j);
    return tot;
}
```

## 7.5 BIT

```
#define lowbit(x) x&-x

int arr[N]; //紀錄前綴和
int bit[N];

void conv(int a[], int n)  //離散化
{
    vector<int> tmp;
    for(int i = 1; i <= n; i++) tmp.push_back(a[i]);
    sort(tmp.begin(), tmp.end());
    for(int i = 1; i <= n; i++) a[i] = lower_bound(tmp.begin(), tmp.end(), a[i]) - tmp.begin
         () + 1;
}

void buildbit() //每個bit[x]紀錄[x-lowbit(x)+1, x]的總和
{
    for(int i = 0; i < n; i++) bit[i] = arr[i]-arr[i-lowbit(i)];
}

int sum(int x) //查詢[1,x]的總和
{
    int rtn = 0;
    for(;x;x-=lowbit(x)) rtn += bit[x];
    return rtn;
}

void modify(int x, int d) //把位置x的東西加上d
{
    for(;x<=n;x+=lowbit(x)) bit[x] += d;
}
```

## 7.6 1d_segTree_tag

```
//線段樹懶人標記：一維陣列區間加值區間乘值區間查詢總和
struct Node //data = data*mul+add;
{
    ll data, mul, add;
};

ll getval(int l, int r, int idx)
{
    return (st[idx].data*st[idx].mul%MD+(r-l+1)*st[idx].add%MD)%MD;
}

void up(int l, int r, int idx)
{
    int mid = l+(r-l)/2;
    st[idx].data = (getval(l, mid, idx*2)+getval(mid+1, r, idx*2+1))%MD;
}

void down(int l, int r, int idx)
{
    st[idx].data = getval(l, r, idx);
    int lson = idx*2, rson = idx*2+1;
    if(l != r)
    {
        st[lson].mul = st[lson].mul*st[idx].mul%MD;
        st[lson].add = (st[lson].add*st[idx].mul+st[idx].add)%MD;
        st[rson].mul = st[rson].mul*st[idx].mul%MD;
        st[rson].add = (st[rson].add*st[idx].mul+st[idx].add)%MD;
    }
    st[idx].mul = 1, st[idx].add = 0;
}

void buildst(int l, int r, int idx)
{
    st[idx].mul = 1, st[idx].add = 0;
    if(l == r)
    {
        st[idx].data = arr[l];
        return;
    }
    int mid = l+(r-l)/2;
    buildst(l, mid, idx*2);
    buildst(mid+1, r, idx*2+1);
    up(l, r, idx);
}

void add(int l, int r, int idx, int L, int R, int v) //操作L,R
{
    if(r < L || R < l) return;
    if(L <= l && r <= R)
    {
        st[idx].add = (st[idx].add+v)%MD;
        return;
    }
    down(l, r, idx);
    int mid = l+(r-l)/2;
    add(l, mid, idx*2, L, R, v);
    add(mid+1, r, idx*2+1, L, R, v);
    up(l, r, idx);
}

void mul(int l, int r, int idx, int L, int R, int v)
{
    if(r < L || R < l) return;
    if(L <= l && r <= R)
    {
```

```
        st[idx].add = st[idx].add*v%MD;
        st[idx].mul = st[idx].mul*v%MD;
        return;
    }
    down(l, r, idx);
    int mid = l+(r-l)/2;
    mul(l, mid, idx*2, L, R, v);
    mul(mid+1, r, idx*2+1, L, R, v);
    up(l, r, idx);
}

ll query(int l, int r, int idx, int L, int R)
{
    if(r < L || R < l) return 0;
    if(L <= l && r <= R)
    {
        return getval(l, r, idx);
    }
    down(l, r, idx);
    int mid = l+(r-l)/2;
    return (query(l, mid, idx*2, L, R)+query(mid+1, r, idx*2+1, L, R))%MD;
}
```

## 7.7 1d_segTree

```
void buildst(int l, int r, int idx)  //l, r是st的區間
{
    if(l == r)
    {
        st[idx] = arr[l];
        return;
    }
    int mid = (l+r)/2;
    buildst(l, mid, idx*2);
    buildst(mid+1, r, idx*2+1);
    st[idx] = max(st[idx*2], st[idx*2+1]);
}

ll query(int l, int r, int idx, int L, int R)  //L,R是操作的區間
{
    if(r < L || R < l) return -INF;
    if(L <= l && r <= R) return st[idx];
    int mid = (l+r)/2;
    return max(query(l, mid, idx*2, L, R), query(mid+1, r, idx*2+1, L, R));
}

void modify(int l, int r, int idx, int x, int v)
{
    if(r < x || x < l) return;
    if(l == r)
    {
        st[idx] += v; return;
    }
    int mid = (l+r)/2;
    modify(l, mid, idx*2, x, v);
    modify(mid+1, r, idx*2+1, x, v);
    st[idx] = max(st[idx*2], st[idx*2+1]);
}
```

# ACM ICPC Team Reference - 燈泡邪教

## Contents