Library

区間制御

グラフ・木

幾何

フロー

動的計画法

データ構造

その他

剰余演算

構文解析

サイコロ

行列

分数

方程式の求解(GaussJordan)

四面体

区間制御

```
struct Chien{
  int n;
  vector<int> dat,laz;
  const int def=0;
  Chien(){}
  Chien(int n_){init(n_);}
  void init(int n_){
    n=1;
    while(n<n_) n*=2;
    dat.clear();
    dat.resize(2*n-1,def);
    laz.clear();
    laz.resize(2*n-1,0);
  }
  inline void eval(int len,int k){
    if(k*2+1<n*2-1){
      laz[k*2+1]+=laz[k];
      laz[k*2+2]+=laz[k];
    }
    dat[k]+=laz[k]*len;
    laz[k]=0;
  }
  int update(int a,int b,int x,int k,int l,int r){
    eval(r-l,k);
    if(r<=a||b<=l) return dat[k]+laz[k]*(r-l);
    if(a<=l&&r<=b) return dat[k]+(laz[k]+=x)*(r-l);
    eval(r-l,k);
    return dat[k]=update(a,b,x,k*2+1,l,(l+r)/2)
      +update(a,b,x,k*2+2,(l+r)/2,r);
  }
  int query(int a,int b,int k,int l,int r){
    eval(r-l,k);
    if(r<=a||b<=l) return def;
    if(a<=l&&r<=b) return dat[k];
    int vl=query(a,b,k*2+1,l,(l+r)/2);
    int vr=query(a,b,k*2+2,(l+r)/2,r);
    return vl+vr;
  }
  int update(int a,int b,int x){
    return update(a,b,x,0,0,n);
  }
  int query(int a,int b){
    return query(a,b,0,0,n);
  }
};


struct RCQ{
  int n;
  vector<vector<int> > dat;
  RCQ(){}
  RCQ(int n_,int* c){init(n_,c);}
  void init(int n_,int *c){
    n=1;
    while(n<n_) n*=2;
    dat.clear();
    dat.resize(2*n-1);
    construct(n_,c);
  }
  void construct(int n_,int *c){
    for(int i=0;i<n_;i++)
      dat[n-1+i].push_back(c[i]);
    for(int i=n-2;i>=0;i--){
      for(int j:dat[i*2+1]) dat[i].push_back(j);
      for(int j:dat[i*2+2]) dat[i].push_back(j);
      sort(dat[i].begin(),dat[i].end());
    }
  }
  int query(int a,int b,int x,int k,int l,int r){
    if(r<=a||b<=l) return 0;
    if(a<=l&&r<=b){
      int res=0;
      auto latte=
        lower_bound(dat[k].begin(),dat[k].end(),x);
      res=dat[k].end()-latte;
      return res;
    }
    int vl=query(a,b,x,k*2+1,l,(l+r)/2);
    int vr=query(a,b,x,k*2+2,(l+r)/2,r);
    return vl+vr;
  }
  int query(int a,int b,int x){
    return query(a,b,x,0,0,n);
  }
};

struct RMQ{
  int n;
  vector<int> dat;
  const int def=INT_MAX;
  RMQ(){}
  RMQ(int n_){init(n_);}
  RMQ(int n_,int* a){init(n_);construct(n_,a);}
  void init(int n_){
    n=1;
    while(n<n_) n*=2;
    dat.clear();
    dat.resize(2*n-1,def);
  }
  void construct(int n_, int* a){
    for(int i=0;i<n_;i++) dat[i+n-1]=a[i];
    for(int i=n-2;i>=0;i--)
      dat[i]=min(dat[i*2+1],dat[i*2+2]);
  }
  void update(int k,int a){
    k+=n-1;
    dat[k]=a;
    while(k>0){
      k=(k-1)/2;
      dat[k]=min(dat[k*2+1],dat[k*2+2]);
    }
  }
  int query(int a,int b,int k,int l,int r){
    if(r<=a||b<=l) return def;
    if(a<=l&&r<=b) return dat[k];
    int vl=query(a,b,k*2+1,l,(l+r)/2);
    int vr=query(a,b,k*2+2,(l+r)/2,r);
    return min(vl,vr);
  }
  int query(int a,int b){
    return query(a,b,0,0,n);
  }
};

struct RUP{
  int n;
  vector<int> dat,laz;
  const int def=INT_MAX;
  RUP(){}
  RUP(int n_){init(n_);}
  void init(int n_){
    n=1;
    while(n<n_) n*=2;
    dat.clear();
    dat.resize(2*n-1,def);
    laz.clear();
    laz.resize(2*n-1,-1);
  }
  inline void eval(int len,int k){
    if(laz[k]<0) return;
    if(k*2+1<n*2-1){
      laz[k*2+1]=laz[k];
      laz[k*2+2]=laz[k];
    }
    dat[k]=laz[k];
    laz[k]=-1;
  }
  void update(int a,int b,int x,int k,int l,int r){
    eval(r-l,k);
    if(r<=a||b<=l) return;
    if(a<=l&&r<=b){
      laz[k]=x;
      return;
    }
    eval(r-l,k);
    update(a,b,x,k*2+1,l,(l+r)/2);
    update(a,b,x,k*2+2,(l+r)/2,r);
  }
  int query(int a,int b,int k,int l,int r){
    eval(r-l,k);
    if(r<=a||b<=l) return def;
    if(a<=l&&r<=b) return dat[k];
    int vl=query(a,b,k*2+1,l,(l+r)/2);
    int vr=query(a,b,k*2+2,(l+r)/2,r);
    return min(vl,vr);
  }
  void update(int a,int b,int x){
    update(a,b,x,0,0,n);
  }
  int query(int a){
    return query(a,a+1,0,0,n);
  }
};

struct StarrySky{
  int n;
  const int def=0;
  vector<int> datm,data;
  StarrySky(){}
  StarrySky(int n_){init(n_);}
  void init(int n_){
    n=1;
  while(n<n_) n*=2;
  datm.clear();
  datm.resize(n*2-1,def);
  data.clear();
  data.resize(n*2-1,0);
 }
 void add(int a,int b,int x,int k,int l,int r){
  if(r<=a||b<=l) return;
  if(a<=l&&r<=b){
   data[k]+=x;
   return;
  }
  add(a,b,x,k*2+1,l,(l+r)/2);
  add(a,b,x,k*2+2,(l+r)/2,r);
  datm[k]=max(datm[k*2+1]+data[k*2+1],
    datm[k*2+2]+data[k*2+2]);
 }
 int query(int a,int b,int k,int l,int r){
  if(r<=a||b<=l) return def;
  if(a<=l&&r<=b) return datm[k]+data[k];
  int vl=query(a,b,k*2+1,l,(l+r)/2);
  int vr=query(a,b,k*2+2,(l+r)/2,r);
  return max(vl,vr)+data[k];
 }
 void add(int a,int b,int x){
  add(a,b,x,0,0,n);
 }
 int query(int a,int b){
  return query(a,b,0,0,n);
 }
};
```

```
struct BIT{
 vector<int> bit;
 int n;
 //1-indexed
 BIT(){init(-1);}
 BIT(int n_){init(n_);}
 void init(int n_){
  n=n_;
  bit.clear();
  bit.resize(n+1,0);
 }
 int sum(int i){
  int s=0;
  while(i>0){
   s+=bit[i];
   i-=i&-i;
  }
  return s;
 }
 void add(int i,int x){
  while(i<=n){
   bit[i]+=x;
   i+=i&-i;
  }
 }
 int sum0(int i){
  return sum(i+1);
 }
 void add0(int i,int x){
  add(i+1,x);
 }
};
```

```
struct AVL{
  struct node{
    int key;
    int size,height;
    node *child[2];
    node(const int &key):key(key),size(1),height(1){
      child[0]=child[1]=0;
    }
  } *root;
  typedef node *pointer;
  AVL(){root=NULL;}
  pointer find(int key){
    return find(root,key);
  }
  node *find(node *t,const int key){
    if(t==NULL) return NULL;
    if(key==(t->key)) return t;
    else if(key<(t->key)) return find(t-
>child[0],key);
    else return find(t->child[1],key);
  }
  void insert(const int key){
    root=insert(root,new node(key));
  }
  node *insert(node *t,node *x){
    if(t==NULL) return x;
    if((x->key)<=(t->key)) t->child[0]=insert(t-
>child[0],x);
    else t->child[1]=insert(t->child[1],x);
    t->size+=1;
    return balance(t);
  }
  void erase(const int key){
    int t=key;
    if(find(t)==NULL) return;
    root=erase(root,key);
  }
  node *erase(node *t,const int x){
    if(t==NULL) return NULL;
    if(x==(t->key)){
      return move_down(t->child[0],t->child[1]);
    }else{
      if(x<(t->key)) t->child[0]=erase(t-
>child[0],x);
      else t->child[1]=erase(t->child[1],x);
      t->size-=1;
      return balance(t);
    }
  }
  node *move_down(node *t,node *rhs){
    if(t==NULL) return rhs;
    t->child[1]=move_down(t->child[1],rhs);
    return balance(t);
  }
  int sz(node *t){
    if(t!=NULL) return t->size;
    return 0;
  }
  int ht(node *t){
    if(t!=NULL) return t->height;
    return 0;
  }
  node *rotate(node *t,int l,int r){
    node *s=t->child[r];
    t->child[r]=s->child[l];
    s->child[l]=balance(t);
    if(t!=NULL) t->size=sz(t->child[0])+sz(t-
>child[1])+1;
    if(s!=NULL) s->size=sz(s->child[0])+sz(s-
>child[1])+1;
    return balance(s);
  }
  node *balance(node *t){
    for(int i=0;i<2;i++){
      if(ht(t->child[!i])-ht(t->child[i])<-1){
    if(ht(t->child[i]->child[!i])-ht(t->child[i]-
>child[i])>0)
      t->child[i]=rotate(t->child[i],i,!i);
    return rotate(t,!i,i);
      }
    }
    if(t!=NULL) t->height=max(ht(t->child[0]),ht(t-
>child[1]))+1;
    if(t!=NULL) t->size=sz(t->child[0])+sz(t-
>child[1])+1;
    return t;
  }
  pointer rank(int k){
    return rank(root,k);
  }
  pointer rank(node *t,int k){
    if(t==NULL) return NULL;
    int m=sz(t->child[0]);
    if(k<m) return rank(t->child[0],k);
    if(k==m) return t;
    if(k>m) return rank(t->child[1],k-m-1);
  }
  int index(int key){
    if(find(key)==NULL) return -1;
    return index(root,key);
  }
  int index(node *t,int key){
    if(key==(t->key)) return sz(t->child[0]);
    if(key<(t->key)) return index(t->child[0],key);
    else return sz(t)-sz(t->child[1])+index(t-
>child[1],key);
  }
};
```

```
struct KDTree{
  class Node{
  public:
    int location;
    int p,l,r;
    Node(){}
  };
  class Point{
  public:
    int id,x,y;
    Point(){}
    Point(int id,int x,int y): id(id),x(x),y(y){}
    bool operator<(const Point &p)const{
      return id<p.id;
    }
    void print(){
      printf("%lld\n",id);
    }
  };

  static const int NIL = -1;

  int N;
  vector<Point> P;
  vector<Node> T;
  int np;

  KDTree(){}
  KDTree(int N){init(N);}

  void init(int N_){
    N=N_;
    P.clear();
    T.clear();
    P.resize(N);
    T.resize(N);
  }
  static bool lessX(const Point &p1,const Point &p2)
{return p1.x<p2.x;}
  static bool lessY(const Point &p1,const Point &p2)
{return p1.y<p2.y;}

  int makeKDTree(int l,int r,int depth){
    if(!(l<r)) return NIL;
    int mid=(l+r)/2;
    int t = np++;
    if(depth%2==0){
      sort(P.begin()+l,P.begin()+r,lessX);
    }else{
      sort(P.begin()+l,P.begin()+r,lessY);
    }
    T[t].location=mid;
    T[t].l=makeKDTree(l,mid,depth+1);
    T[t].r=makeKDTree(mid+1,r,depth+1);
    return t;
  }

  void find(int v,int sx,int tx,int sy,int ty,int
depth,vector<Point> &ans){
    int x=P[T[v].location].x;
    int y=P[T[v].location].y;
    if(sx<=x&&x<=tx&&sy<=y&&y<=ty){
      ans.push_back(P[T[v].location]);
    }
    if(depth%2==0){
      if(T[v].l!=NIL){
    if(sx<=x) find(T[v].l,sx,tx,sy,ty,depth+1,ans);
      }
      if(T[v].r!=NIL){
    if(x<=tx) find(T[v].r,sx,tx,sy,ty,depth+1,ans);
      }
    }else{
      if(T[v].l!=NIL){
    if(sy<=y) find(T[v].l,sx,tx,sy,ty,depth+1,ans);
      }
      if(T[v].r!=NIL){
    if(y<=ty) find(T[v].r,sx,tx,sy,ty,depth+1,ans);
      }
    }
  }
};

struct SparseTable{
  int n,h;
  vector<vector<int> > dat;
  vector<int> ht;
  SparseTable(){}
  SparseTable(int n_,int *arr){init(n_,arr);}
  void init(int n_,int *arr){
    n=1;h=1;
    while(n<n_) n*=2,h++;
    dat.clear();
    dat.resize(h,vector<int>(n_));
    for(int j=0;j<n_;j++) dat[0][j]=arr[j];
    for(int i=1,p=1;i<h;i++){
      for(int j=0;j<n_;j++){
    dat[i][j]=dat[i-1][j];
    if(j+p<n_) dat[i][j]=min(dat[i][j],dat[i-1]
[j+p]);
      }
      p*=2;
    }
    ht.resize(n_);
    ht[0]=ht[1]=0;
    for(int j=2;j<n_;j++)
      ht[j]=ht[j>>1]+1;
  }
  int query(int a,int b){
    b--;//[a,b)->[a,b-1]
    int l=b-a+1;
    return min(dat[ht[l]][a],dat[ht[l]][b-(1<<ht[l])
+1]);
  }
};
```

グラフ・木

```
#define MAX_V 11111
typedef pair<int,int> P;
vector<P> G[MAX_V];
bool used[MAX_V];
int prim(){
  int res=0;
  priority_queue<P,vector<P>,greater<P> > q;
  q.push(P(0,0));
  memset(used,0,sizeof(used));
  while(!q.empty()){
    P p=q.top();q.pop();
    int v=p.second,d=p.first;
    if(used[v]) continue;
    used[v]=1;
    res+=d;
    for(int i=0;i<(int)G[v].size();i++){
      q.push(G[v][i]);
    }
  }
  return res;
}

struct UnionFind{};
struct edge{
  int from,to,cost;
  edge(){}
  edge(int from,int to,int
cost):from(from),to(to),cost(cost){}
  bool operator<(const edge& e) const{
    return cost<e.cost;
  }
};
int kruskal(int N,vector<edge> edges){
  int res=0;
  sort(edges.begin(),edges.end());
  UnionFind uf(N+1);
  for(int i=0;i<(int)edges.size();i++){
    edge e=edges[i];
    if(!uf.same(e.from,e.to)){
      res+=e.cost;
      uf.unite(e.from,e.to);
    }
  }
  return res;
}


struct LowestCommonAncestor{
  const int MAX_LOG_V = 50;

  vector<vector<int> > G,parent;
  int root=0,V;
  vector<int> depth;
  LowestCommonAncestor(){}
  LowestCommonAncestor(int V):V(V){init();}

  void init(){
    for(int i=0;i<(int)G.size();i++) G[i].clear();
    G.clear();
    for(int i=0;i<(int)parent.size();i++)
parent[i].clear();
    parent.clear();
    depth.clear();
    G.resize(V);
    parent.resize(MAX_LOG_V,vector<int>(V));
  depth.resize(V);
}

void add_edge(int u,int v){
  G[u].push_back(v);
  G[v].push_back(u);
}

void dfs(int v,int p,int d){
  parent[0][v]=p;
  depth[v]=d;
  for(int i=0;i<(int)G[v].size();i++){
    if(G[v][i]!=p) dfs(G[v][i],v,d+1);
  }
}

void construct(){
  dfs(root,-1,0);
  for(int k=0;k+1<MAX_LOG_V;k++){
    for(int v=0;v<V;v++){
   if(parent[k][v]<0) parent[k+1][v]=-1;
   else parent[k+1][v]=parent[k][parent[k][v]];
    }
  }
}

int lca(int u,int v){
  if(depth[u]>depth[v]) swap(u,v);
  for(int k=0;k<MAX_LOG_V;k++){
    if((depth[v]-depth[u])>>k&1){
   v=parent[k][v];
    }
  }
  if(u==v) return u;
  for(int k=MAX_LOG_V-1;k>=0;k--){
    if(parent[k][u]!=parent[k][v]){
   u=parent[k][u];
```

```
     v=parent[k][v];
      }
    }
    return parent[0][u];
  }
};

struct BiconectedGraph{
  typedef pair<int,int> P;
  vector<vector<int> > G,C,T;
  vector<int> ord,low,belong;
  vector<P> B;
  int V;
  BiconectedGraph(){}
  BiconectedGraph(int n){
    G.clear();
    C.clear();
    T.clear();
    G.resize(n);
    C.resize(n);
    T.resize(n);
  }
  bool is_bridge(int u,int v){
    if(ord[u]>ord[v]) swap(u,v);
    return ord[u]<low[v];
  }
  void dfs(int u,int p,int &k){
    ord[u]=low[u]=k;
    ++k;
    for(int v:G[u]){
      if(v==p) continue;
      if(ord[v]>=0){
     low[u]=min(low[u],ord[v]);
      }else{
     dfs(v,u,k);
     low[u]=min(low[u],low[v]);
      }
    if(is_bridge(u,v)) B.push_back(P(u,v));
  }
}
void fill_component(int c,int u){
  C[c].push_back(u);
  belong[u]=c;
  for(int v:G[u]){
    if(belong[v]>=0||is_bridge(u,v)) continue;
    fill_component(c,v);
  }
}
void add_component(int u,int &k){
  if(belong[u]>=0) return;
  fill_component(k++,u);
}

void biconnectedgraph(int n){
  int k=0;
  ord.clear();
  ord.resize(n,-1);
  low.clear();
  low.resize(n);
  belong.clear();
  belong.resize(n,-1);
  for(int u=0;u<n;u++){
    if(ord[u]>=0) continue;
    dfs(u,-1,k);
  }
  k=0;
  for(int i=0;i<(int)B.size();i++){
    add_component(B[i].first,k);
    add_component(B[i].second,k);
  }
  add_component(0,k);
  V=k;
  for(int i=0;i<(int)B.size();i++){
```

```
      int
u=belong[B[i].first],v=belong[B[i].second];
      T[u].push_back(v);
      T[v].push_back(u);
    }
  }
};

struct HLDecomposition {
  vector<vector<int>> g;

  vector<int> vid, head, heavy, parent, depth, inv;

  HLDecomposition(){}
  HLDecomposition(int n){init(n);}

  void init(int n){
    for(auto &a:g) a.clear();
    g.clear();
    vid.clear();
    head.clear();
    heavy.clear();
    parent.clear();
    depth.clear();
    inv.clear();

    g.resize(n);
    vid.resize(n, -1);
    head.resize(n);
    heavy.resize(n, -1);
    parent.resize(n);
    depth.resize(n);
    inv.resize(n);
  }

  void add_edge(int u, int v) {
    g[u].push_back(v);
  g[v].push_back(u);
}

void build() {
  dfs(0, -1);
  bfs();
}

typedef tuple<int,int,int,int,int,int> T;
int dfs(int curr, int prev) {
  stack<T> st;
  int result;
  int sub, max_sub, i, next;
ENTRYPOINT:
  parent[curr] = prev;
  sub = 1;
  max_sub = 0;
  for(i=0;i<(int)g[curr].size();i++){
    next = g[curr][i];
    if (next != prev) {
   depth[next] = depth[curr] + 1;
   {
     st.emplace(curr,prev,sub,max_sub,i,next);
     prev=curr;curr=next;
     goto ENTRYPOINT;
   }
    RETURNPOINT:
   T t=st.top();st.pop();
   curr    = get<0>(t);
   prev    = get<1>(t);
   sub     = get<2>(t);
   max_sub = get<3>(t);
   i       = get<4>(t);
   next    = get<5>(t);

   int sub_next=result;
   sub += sub_next;
```

```
     if (max_sub < sub_next) max_sub = sub_next,
heavy[curr] = next;
      }
    }
    while(!st.empty()){
      result=sub;
      goto RETURNPOINT;
    }
    return sub;
  }

  void bfs() {
    int k = 0;
    queue<int> q({ 0 });
    while (!q.empty()) {
      int h = q.front(); q.pop();
      for (int i = h; i != -1; i = heavy[i]) {
     vid[i] = k++;
     inv[vid[i]] = i;
     head[i] = h;
     for (int j : g[i]) if (j != parent[i] && j !=
heavy[i]) q.push(j);
      }
    }
  }

  // for_each(vertex)
  // [l,r] <- attention!!
  void for_each(int u, int v, const
function<void(int, int)>& f) {
    if (vid[u] > vid[v]) swap(u, v);
    f(max(vid[head[v]], vid[u]), vid[v]);
    if (head[u] != head[v]) for_each(u,
parent[head[v]], f);
  }

  // for_each(edge)
  // [l,r] <- attention!!
  void for_each_edge(int u, int v, const
function<void(int, int)>& f) {
    if (vid[u] > vid[v]) swap(u, v);
    if (head[u] != head[v]){
      f(vid[head[v]], vid[v]);
      for_each_edge(u, parent[head[v]], f);
    }else{
      if(u!=v) f(vid[u]+1,vid[v]);
    }
  }

  int lca(int u,int v){
    if(vid[u]>vid[v]) swap(u,v);
    if(head[u]==head[v]) return u;
    return lca(u,parent[head[v]]);
  }

  int distance(int u,int v){
    return depth[u]+depth[v]-2*depth[lca(u,v)];
  }
};
#define MAX 11
bool e[MAX][MAX];
int mod_pow(int x,int n,int mod){
  int res=1;
  while(n>0){
    if(n&1) (res*=x)%=mod;
    (x*=x)%=mod;
    n>>=1;
  }
  return res;
}
int I[1<<MAX],bn[1<<MAX];
bool check(int n,int k){
  int g=0;
  int MOD=10009;
```

```
  for(int i=0;i<(1<<n);i++){
    if(bn[i]%2==0) g+=mod_pow(I[i],k,MOD);
    else g-=mod_pow(I[i],k,MOD);
  }
  return (g%MOD+MOD)%MOD!=0;
}
int paint(int n){
  vector<int> N(1<<n);
  for(int i=0;i<n;i++){
    int bit=(1<<i);
    for(int j=0;j<n;j++)
      if(e[i][j]) bit|=(1<<j);
    N[i]=bit;
  }
  memset(I,0,sizeof(I));
  I[0]=1;
  for(int S=1;S<(1<<n);S++){
    bn[S]=__builtin_popcountll(S);
    int v=0;
    while(!((S>>v)&1)) v++;
    I[S]=I[S-(1<<v)]+I[S&~N[v]];
  }
  int l=0,r=n;
  while(l+1<r){
    int m=(l+r)/2;
    if(check(n,m)) r=m;
    else l=m;
  }
  return r;
}
```

```
struct SCC{
  int V;
  vector<vector<int> > G,rG,T;
  vector<int> vs,used,cmp;
  SCC(){}
  SCC(int V):V(V){init();}
  void init(){
    G.clear();
    rG.clear();
    vs.clear();
    used.clear();
    cmp.clear();
    T.clear();

    G.resize(V);
    rG.resize(V);
    used.resize(V);
    cmp.resize(V);
  }
  void add_edge(int from,int to){
    G[from].push_back(to);
    rG[to].push_back(from);
  }
  void dfs(int v){
    used[v]=1;
    for(int i=0;i<(int)G[v].size();i++){
      if(!used[G[v][i]]) dfs(G[v][i]);
    }
    vs.push_back(v);
  }
  void rdfs(int v,int k){
    used[v]=1;
    cmp[v]=k;
    T[k].push_back(v);
    for(int i=0;i<(int)rG[v].size();i++){
      if(!used[rG[v][i]]) rdfs(rG[v][i],k);
    }
  }

  int scc(){
    fill(used.begin(),used.end(),0);
    vs.clear();
    for(int v=0;v<V;v++){
      if(!used[v]) dfs(v);
    }
    fill(used.begin(),used.end(),0);
    int k=0;
    for(int i=vs.size()-1;i>=0;i--){
      if(!used[vs[i]]){
     T.push_back(vector<int>());
     rdfs(vs[i],k++);
      }
    }
    return k;
  }
};

struct TopologicalSort{
  int n;
  vector<set<int> > G;
  vector<int> indeg,V,p;
  TopologicalSort(){}
  TopologicalSort(int n):n(n){init();}

  void init(){
    for(int i=0;i<(int)G.size();i++) G[i].clear();
    G.clear();
    indeg.clear();
    V.clear();
    p.clear();
    G.resize(n);
    indeg.resize(n);
    V.resize(n);
  }

  void add_edge(int s,int t){
    G[s].insert(t);
  }

  void bfs(int s){
    queue<int> q;
    q.push(s);
    V[s]=1;
    while(!q.empty()){
      int u=q.front();q.pop();
      p.push_back(u);
      for(int v:G[u]){
     indeg[v]--;
     if(indeg[v]==0&&!V[v]){
       V[v]=1;
       q.push(v);
     }
      }
    }
  }

  void tsort(){
    fill(V.begin(),V.end(),0);
    fill(indeg.begin(),indeg.end(),0);
    for(int u=0;u<n;u++)
      for(int v:G[u])
     indeg[v]++;
    for(int u=0;u<n;u++)
      if(indeg[u]==0&&!V[u]) bfs(u);
    for(int i=0;i<n;i++)
      cout<<p[i]<<endl;
  }
};
```

```
vector <int> treeconstruction(vector <int> d) {
  const int MAX = 111;
  vector<int> res,NG;
  int n=d.size();
  int m=0;
  int t[MAX]={};
  queue<int> q[MAX];
  for(int i=0;i<n;i++){
    t[d[i]]++;
    q[d[i]].push(i);
    m=max(m,d[i]);
  }
  bool f=1;
  for(int i=0;i<(m+1)/2;i++) f&=t[m-i]>=2;
  if(m%2) f&=t[m/2+1]==2;
  else f&=t[m/2]==1;
  for(int i=0;i<n;i++){
    if(m%2) f&=d[i]>=m/2+1;
    else f&=d[i]>=m/2;
  }
  if(!f) return NG;

  bool used[MAX]={};
  int b[MAX][2]={};
  if(m%2){
    for(int i=0;i<=m/2;i++){
      b[m-i][0]=q[m-i].front();q[m-i].pop();
      b[m-i][1]=q[m-i].front();q[m-i].pop();
      used[b[m-i][0]]=1;
      used[b[m-i][1]]=1;
      if(i){
     res.push_back(b[m-i+1][0]);
     res.push_back(b[m-i][0]);
     res.push_back(b[m-i+1][1]);
     res.push_back(b[m-i][1]);
      }
      if(i==m/2){
```

```
     res.push_back(b[m-i][0]);
     res.push_back(b[m-i][1]);
      }
    }
    for(int i=0;i<n;i++){
      if(used[i]) continue;
      res.push_back(i);
      res.push_back(b[d[i]-1][0]);
    }
  }else{
    for(int i=0;i<m/2;i++){
      b[m-i][0]=q[m-i].front();q[m-i].pop();
      b[m-i][1]=q[m-i].front();q[m-i].pop();
      used[b[m-i][0]]=1;
      used[b[m-i][1]]=1;
      if(i){
     res.push_back(b[m-i+1][0]);
     res.push_back(b[m-i][0]);
     res.push_back(b[m-i+1][1]);
     res.push_back(b[m-i][1]);
      }
    }
    b[m/2][0]=q[m/2].front();q[m/2].pop();
    used[b[m/2][0]]=1;
    res.push_back(b[m/2+1][0]);
    res.push_back(b[m/2][0]);
    res.push_back(b[m/2+1][1]);
    res.push_back(b[m/2][0]);

    for(int i=0;i<n;i++){
      if(used[i]) continue;
      res.push_back(i);
      res.push_back(b[d[i]-1][0]);
    }
  }
  return res;
}
```

幾何(2D)

```
#define EPS (1e-10)
#define equals(a,b) (fabs((a)-(b)) < EPS)
#define PI 3.141592653589793238

// COUNTER CLOCKWISE
static const int CCW_COUNTER_CLOCKWISE = 1;
static const int CCW_CLOCKWISE = -1;
static const int CCW_ONLINE_BACK = 2;
static const int CCW_ONLINE_FRONT = -2;
static const int CCW_ON_SEGMENT = 0;

//Intercsect Circle & Circle
static const int ICC_SEPERATE = 4;
static const int ICC_CIRCUMSCRIBE = 3;
static const int ICC_INTERSECT = 2;
static const int ICC_INSCRIBE = 1;
static const int ICC_CONTAIN = 0;

struct Point{
  double x,y;
  Point(){}
  Point(double x,double y) :x(x),y(y){}
  Point operator+(Point p) {return 
Point(x+p.x,y+p.y);}
  Point operator-(Point p) {return Point(x-p.x,y-
p.y);}
  Point operator*(double k){return Point(x*k,y*k);}
  Point operator/(double k){return Point(x/k,y/k);}
  double norm(){return x*x+y*y;}
  double abs(){return sqrt(norm());}

  bool operator < (const Point &p) const{
    return x!=p.x?x<p.x:y<p.y;
  }

  bool operator == (const Point &p) const{
    return fabs(x-p.x)<EPS && fabs(y-p.y)<EPS;
  }
};

istream &operator >> (istream &is,Point &p){
  is>>p.x>>p.y;
  return is;
}

ostream &operator << (ostream &os,Point p){
  os<<fixed<<setprecision(12)<<p.x<<" "<<p.y;
  return os;
}

bool sort_x(Point a,Point b){
  return a.x!=b.x?a.x<b.x:a.y<b.y;
}

bool sort_y(Point a,Point b){
  return a.y!=b.y?a.y<b.y:a.x<b.x;
}

typedef Point Vector;
typedef vector<Point> Polygon;

struct Segment{
  Point p1,p2;
  Segment(){}
  Segment(Point p1, Point p2):p1(p1),p2(p2){}
};
typedef Segment Line;

istream &operator >> (istream &is,Segment &s){
  is>>s.p1>>s.p2;
  return is;
}

struct Circle{
  Point c;
  double r;
  Circle(){}
  Circle(Point c,double r):c(c),r(r){}
};

istream &operator >> (istream &is,Circle &c){
  is>>c.c>>c.r;
  return is;
}

double norm(Vector a){
  return a.x*a.x+a.y*a.y;
}
double abs(Vector a){
  return sqrt(norm(a));
}
double dot(Vector a,Vector b){
  return a.x*b.x+a.y*b.y;
}
double cross(Vector a,Vector b){
  return a.x*b.y-a.y*b.x;
}

bool isOrthogonal(Vector a,Vector b){
  return equals(dot(a,b),0.0);
}

bool isOrthogonal(Point a1,Point a2,Point b1,Point
b2){
  return isOrthogonal(a1-a2,b1-b2);
}
```

```
bool isOrthogonal(Segment s1,Segment s2){
  return equals(dot(s1.p2-s1.p1,s2.p2-s2.p1),0.0);
}

bool isParallel(Vector a,Vector b){
  return equals(cross(a,b),0.0);
}

bool isParallel(Point a1,Point a2,Point b1,Point b2)
{
  return isParallel(a1-a2,b1-b2);
}

bool isParallel(Segment s1,Segment s2){
  return equals(cross(s1.p2-s1.p1,s2.p2-s2.p1),0.0);
}

Point project(Segment s,Point p){
  Vector base=s.p2-s.p1;
  double r=dot(p-s.p1,base)/norm(base);
  return s.p1+base*r;
}

Point reflect(Segment s,Point p){
  return p+(project(s,p)-p)*2.0;
}

double arg(Vector p){
  return atan2(p.y,p.x);
}

Vector polar(double a,double r){
  return Point(cos(r)*a,sin(r)*a);
}

int ccw(Point p0,Point p1,Point p2);
```

```
bool intersectSS(Point p1,Point p2,Point p3,Point
p4);
bool intersectSS(Segment s1,Segment s2);
int intersectCC(Circle c1,Circle c2);
bool intersectSC(Segment s,Circle c);
double getDistanceLP(Line l,Point p);
double getDistanceSP(Segment s,Point p);
double getDistanceSS(Segment s1,Segment s2);
Point getCrossPointSS(Segment s1,Segment s2);
Point getCrossPointLL(Line l1,Line l2);
pair<Point,Point> getCrossPointCL(Circle c,Line l);
pair<Point,Point> getCrossPointCC(Circle c1,Circle
c2);
int contains(Polygon g,Point p);
Polygon andrewScan(Polygon s);
Polygon convex_hull(Polygon ps);
double diameter(Polygon s);
bool isConvex(Polygon p);
double area(Polygon s);
Polygon convexCut(Polygon p,Line l);
Line bisector(Point p1,Point p2);
Vector translate(Vector v,double theta);
vector<Line> corner(Line l1,Line l2);

int ccw(Point p0,Point p1,Point p2){
  Vector a = p1-p0;
  Vector b = p2-p0;
  if(cross(a,b) > EPS) return CCW_COUNTER_CLOCKWISE;
  if(cross(a,b) < -EPS) return CCW_CLOCKWISE;
  if(dot(a,b) < -EPS) return CCW_ONLINE_BACK;
  if(a.norm()<b.norm()) return CCW_ONLINE_FRONT;
  return CCW_ON_SEGMENT;
}

bool intersectSS(Point p1,Point p2,Point p3,Point
p4){
  return (ccw(p1,p2,p3)*ccw(p1,p2,p4) <= 0 &&
       ccw(p3,p4,p1)*ccw(p3,p4,p2) <= 0 );
}

bool intersectSS(Segment s1,Segment s2){
  return intersectSS(s1.p1,s1.p2,s2.p1,s2.p2);
}

int intersectCC(Circle c1,Circle c2){
  if(c1.r<c2.r) swap(c1,c2);
  double d=abs(c1.c-c2.c);
  double r=c1.r+c2.r;
  if(equals(d,r)) return ICC_CIRCUMSCRIBE;
  if(d>r) return ICC_SEPERATE;
  if(equals(d+c2.r,c1.r)) return ICC_INSCRIBE;
  if(d+c2.r<c1.r) return ICC_CONTAIN;
  return ICC_INTERSECT;
}

bool intersectSC(Segment s,Circle c){
  double d=getDistanceSP(s,c.c);
  return d<=c.r;
}

double getDistanceLP(Line l,Point p){
  return abs(cross(l.p2-l.p1,p-l.p1)/abs(l.p2-
l.p1));
}

double getDistanceSP(Segment s,Point p){
  if(dot(s.p2-s.p1,p-s.p1) < 0.0 ) return abs(p-
s.p1);
  if(dot(s.p1-s.p2,p-s.p2) < 0.0 ) return abs(p-
s.p2);
  return getDistanceLP(s,p);
}

double getDistanceSS(Segment s1,Segment s2){
  if(intersectSS(s1,s2)) return 0.0;
  return
min(min(getDistanceSP(s1,s2.p1),getDistanceSP(s1,s2.
p2)),

min(getDistanceSP(s2,s1.p1),getDistanceSP(s2,s1.p2))
);
}

Point getCrossPointSS(Segment s1,Segment s2){
  Vector base=s2.p2-s2.p1;
  double d1=abs(cross(base,s1.p1-s2.p1));
  double d2=abs(cross(base,s1.p2-s2.p1));
  double t=d1/(d1+d2);
  return s1.p1+(s1.p2-s1.p1)*t;
}

Point getCrossPointLL(Line l1,Line l2){
  double a=cross(l1.p2-l1.p1,l2.p2-l2.p1);
  double b=cross(l1.p2-l1.p1,l1.p2-l2.p1);
  if(abs(a)<EPS&&abs(b)<EPS) return l2.p1;
  return l2.p1+(l2.p2-l2.p1)*(b/a);
}

pair<Point,Point> getCrossPointCL(Circle c,Line l){
  Vector pr=project(l,c.c);
  Vector e=(l.p2-l.p1)/abs(l.p2-l.p1);
  double base=sqrt(c.r*c.r-norm(pr-c.c));
  return make_pair(pr+e*base,pr-e*base);
}

pair<Point,Point> getCrossPointCC(Circle c1,Circle
c2){
  double d=abs(c1.c-c2.c);
  double a=acos((c1.r*c1.r+d*d-c2.r*c2.r)/
(2*c1.r*d));
  double t=arg(c2.c-c1.c);
  return
make_pair(c1.c+polar(c1.r,t+a),c1.c+polar(c1.r,t-
a));
}

// IN:2 ON:1 OUT:0
int contains(Polygon g,Point p){
  int n=g.size();
  bool x=false;
  for(int i=0;i<n;i++){
    Point a=g[i]-p,b=g[(i+1)%n]-p;
    if(fabs(cross(a,b)) < EPS && dot(a,b) < EPS)
return 1;
    if(a.y>b.y) swap(a,b);
    if(a.y < EPS && EPS < b.y && cross(a,b) > EPS )
x = !x;
  }
  return (x?2:0);
}

Polygon andrewScan(Polygon s){
  Polygon u,l;
  if(s.size()<3) return s;
  sort(s.begin(),s.end());
  u.push_back(s[0]);
  u.push_back(s[1]);
  l.push_back(s[s.size()-1]);
  l.push_back(s[s.size()-2]);
  for(int i=2;i<(int)s.size();i++){
    for(int
n=u.size();n>=2&&ccw(u[n-2],u[n-1],s[i])!
=CCW_CLOCKWISE;n--){
      u.pop_back();
    }
    u.push_back(s[i]);
  }
  for(int i=s.size()-3;i>=0;i--){
    for(int
n=l.size();n>=2&&ccw(l[n-2],l[n-1],s[i])!
=CCW_CLOCKWISE;n--){
      l.pop_back();
    }
    l.push_back(s[i]);
  }
  reverse(l.begin(),l.end());
  for(int i=u.size()-2;i>=1;i--) l.push_back(u[i]);
  return l;
}

Polygon convex_hull(Polygon ps){
  int n=ps.size();
  sort(ps.begin(),ps.end(),sort_y);
  int k=0;
  Polygon qs(n*2);
  for(int i=0;i<n;i++){
    while(k>1&&cross(qs[k-1]-qs[k-2],ps[i]-
qs[k-1])<0) k--;
    qs[k++]=ps[i];
  }
  for(int i=n-2,t=k;i>=0;i--){
    while(k>t&&cross(qs[k-1]-qs[k-2],ps[i]-
qs[k-1])<0) k--;
    qs[k++]=ps[i];
  }
  qs.resize(k-1);
  return qs;
}

double diameter(Polygon s){
  Polygon p=s;
  int n=p.size();
  if(n==2) return abs(p[0]-p[1]);
  int i=0,j=0;
  for(int k=0;k<n;k++){
    if(p[i]<p[k]) i=k;
    if(!(p[j]<p[k])) j=k;
  }
  double res=0;
  int si=i,sj=j;
  while(i!=sj||j!=si){
    res=max(res,abs(p[i]-p[j]));
    if(cross(p[(i+1)%n]-p[i],p[(j+1)%n]-p[j])<0.0){
      i=(i+1)%n;
    }else{
      j=(j+1)%n;
    }
  }
  return res;
}

bool isConvex(Polygon p){
  bool f=1;
  int n=p.size();
  for(int i=0;i<n;i++){
    int t=ccw(p[(i+n-1)%n],p[i],p[(i+1)%n]);
    f&=t!=CCW_CLOCKWISE;
  }
  return f;
}

double area(Polygon s){
  double res=0;
  for(int i=0;i<(int)s.size();i++){
    res+=cross(s[i],s[(i+1)%s.size()])/2.0;
  }
  return abs(res);
}

Polygon convexCut(Polygon p,Line l){
  Polygon q;
  for(int i=0;i<(int)p.size();i++){
    Point a=p[i],b=p[(i+1)%p.size()];
    if(ccw(l.p1,l.p2,a)!=-1) q.push_back(a);
    if(ccw(l.p1,l.p2,a)*ccw(l.p1,l.p2,b)<0)
      q.push_back(getCrossPointLL(Line(a,b),l));
  }
  return q;
}

Line bisector(Point p1,Point p2){
  Circle c1=Circle(p1,abs(p1-
p2)),c2=Circle(p2,abs(p1-p2));
  pair<Point,Point> p=getCrossPointCC(c1,c2);
  if(cross(p2-p1,p.first-p1)>0)
swap(p.first,p.second);
  return Line(p.first,p.second);
}

Vector translate(Vector v,double theta){
  Vector res;
  res.x=cos(theta)*v.x-sin(theta)*v.y;
  res.y=sin(theta)*v.x+cos(theta)*v.y;
  return res;
}

vector<Line> corner(Line l1,Line l2){
  vector<Line> res;
  if(isParallel(l1,l2)){
    double d=getDistanceLP(l1,l2.p1)/2.0;
    Vector v1=l1.p2-l1.p1;
    v1=v1/v1.abs()*d;
    Point p=l2.p1+translate(v1,90.0*(PI/180.0));
    double d1=getDistanceLP(l1,p);
    double d2=getDistanceLP(l2,p);
    if(abs(d1-d2)>d){
      p=l2.p1+translate(v1,-90.0*(PI/180.0));
    }
    res.push_back(Line(p,p+v1));
  }else{
    Point p=getCrossPointLL(l1,l2);
    Vector v1=l1.p2-l1.p1,v2=l2.p2-l2.p1;
    v1=v1/v1.abs();
    v2=v2/v2.abs();
    res.push_back(Line(p,p+(v1+v2)));
    res.push_back(Line(p,p+translate(v1+v2,90.0*(PI/
180.0))));
  }
  return res;
}
```

幾何(3D)

```
#define EPS (1e-10)
#define equals(a,b) (fabs((a)-(b)) < EPS)
#define PI 3.141592653589793238
struct Point3D{
  double x,y,z;
  Point3D(){}
  Point3D(double x,double y,double z):x(x),y(y),z(z)
{}
  Point3D operator+(Point3D p) {return
Point3D(x+p.x,y+p.y,z+p.z);}
  Point3D operator-(Point3D p) {return Point3D(x-
p.x,y-p.y,z-p.z);}
  Point3D operator*(double k){return
Point3D(x*k,y*k,z*k);}
  Point3D operator/(double k){return Point3D(x/k,y/
k,z/k);}
  double norm(){return x*x+y*y+z*z;}
  double abs(){return sqrt(norm());}
  bool operator < (const Point3D &p) const{
    if(x!=p.x) return x<p.x;
    if(y!=p.y) return y<p.y;
    return z<p.z;
  }
  bool operator == (const Point3D &p) const{
    return fabs(x-p.x)<EPS && fabs(y-p.y)<EPS &&
fabs(z-p.z)<EPS;
  }
};
istream &operator >> (istream &is,Point3D &p){
  is>>p.x>>p.y>>p.z;
  return is;
}
ostream &operator << (ostream &os,Point3D p){
  os<<fixed<<setprecision(12)<<p.x<<" "<<p.y<<"
"<<p.z;
  return os;
}

typedef Point3D Vector3D;
typedef vector<Point3D> Polygon3D;

struct Segment3D{
  Point3D p1,p2;
  Segment3D(){}
  Segment3D(Point3D p1, Point3D p2):p1(p1),p2(p2){}
};
typedef Segment3D Line3D;

istream &operator >> (istream &is,Segment3D &s){
  is>>s.p1>>s.p2;
  return is;
}

struct Sphere{
  Point3D c;
  double r;
  Sphere(){}
  Sphere(Point3D c,double r):c(c),r(r){}
};

istream &operator >> (istream &is,Sphere &c){
  is>>c.c>>c.r;
  return is;
}

double norm(Vector3D a){
  return a.x*a.x+a.y*a.y+a.z*a.z;
}
double abs(Vector3D a){
  return sqrt(norm(a));
}
double dot(Vector3D a,Vector3D b){
  return a.x*b.x+a.y*b.y+a.z*b.z;
}
Vector3D cross(Vector3D a,Vector3D b){
  return Vector3D(a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x);
}

Point3D project(Line3D l,Point3D p){
  Point3D b=l.p2-l.p1;
  double t=dot(p-l.p1,b)/norm(b);
  return l.p1+b*t;
}

Point3D reflect(Line3D l,Point3D p){
  return p+(project(l,p)-p)*2.0;
}

double getDistanceLP(Line3D l,Point3D p){
  return abs(cross(l.p2-l.p1,p-l.p1)/abs(l.p2-l.p1));
}

double getDistanceSP(Segment3D s,Point3D p){
  if(dot(s.p2-s.p1,p-s.p1) < 0.0 ) return abs(p-s.p1);
  if(dot(s.p1-s.p2,p-s.p2) < 0.0 ) return abs(p-s.p2);
  return getDistanceLP(s,p);
}

bool intersectSC(Segment3D s,Sphere c){
  double d=getDistanceSP(s,c.c);
  if(d>c.r) return 0;
  return !((abs(s.p1-c.c)<=c.r)&&(abs(s.p2-c.c)<=c.r));
}
```

フロー

```
struct BipartiteMatching{
  vector<vector<int> > G;
  int V;
  vector<int> match,used;

  BipartiteMatching(){}
  BipartiteMatching(int V):V(V){init();}

  void init(){
    for(int i=0;i<(int)G.size();i++) G[i].clear();
    G.clear();
    match.clear();
    used.clear();
    G.resize(V);
    match.resize(V);
    used.resize(V);
  }
  void add_edge(int u,int v){
    G[u].push_back(v);
    G[v].push_back(u);
  }

  bool dfs(int v){
    used[v]=true;
    for(int i=0;i<(int)G[v].size();i++){
      int u=G[v][i],w=match[u];
      if(w<0||(!used[w]&&dfs(w))){
     match[v]=u;
     match[u]=v;
     return true;
      }
    }
    return false;
  }

  int bipartite_matching(){
    int res=0;
    fill(match.begin(),match.end(),-1);
    for(int v=0;v<V;v++){
      if(match[v]<0){
     fill(used.begin(),used.end(),0);
     if(dfs(v)){
       res++;
     }
      }
    }
    return res;
  }
};
```

```
struct Dinic{
  const int INF=1<<28;

  struct edge {
    int to,cap,rev;
    edge(){}
    edge(int to,int cap,int
rev):to(to),cap(cap),rev(rev){}
  };

  vector<vector<edge> > G;
  vector<map<int,int> > M;
  vector<int> level,iter;

  Dinic(){}
  Dinic(int V){init(V);}

  void init(int V){
    for(int i=0;i<(int)G.size();i++) G[i].clear();
    G.clear();
    for(int i=0;i<(int)M.size();i++) M[i].clear();
    M.clear();
    level.clear();
    iter.clear();
    G.resize(V);
    M.resize(V);
    level.resize(V);
    iter.resize(V);
  }

  void add_edge(int from,int to,int cap){
    M[from][to]=G[from].size();
    M[to][from]=G[to].size();
    G[from].push_back(edge(to,cap,G[to].size()));
    // undirected
    //
G[to].push_back(edge(from,cap,G[from].size()-1));
```

```
  // directed
  G[to].push_back(edge(from,0,G[from].size()-1));
}

void bfs(int s){
  fill(level.begin(),level.end(),-1);
  queue<int> que;
  level[s]=0;
  que.push(s);
  while(!que.empty()){
    int v=que.front();que.pop();
    for(int i=0;i<(int)G[v].size();i++){
   edge &e = G[v][i];
   if(e.cap>0&&level[e.to]<0){
     level[e.to]=level[v]+1;
     que.push(e.to);
   }
    }
  }
}

int dfs(int v,int t,int f){
  if(v==t) return f;
  for(int &i=iter[v];i<(int)G[v].size();i++){
    edge &e=G[v][i];
    if(e.cap>0&&level[v]<level[e.to]){
   int d = dfs(e.to,t,min(f,e.cap));
   if(d>0){
     e.cap-=d;
     G[e.to][e.rev].cap+=d;
     return d;
   }
    }
  }
  return 0;
}
```

```
  int max_flow(int s,int t,int lim){
    int flow=0;
    for(;;){
      bfs(s);
      if(level[t]<0||lim==0) return flow;
      fill(iter.begin(),iter.end(),0);
      int f;
      while((f=dfs(s,t,lim))>0){
     flow+=f;
     lim-=f;
      }
    }
  }

  int max_flow(int s,int t){
    return max_flow(s,t,INF);
  }

  //cap==1 only
  bool back_edge(int s,int t,int from, int to){
    for(int i=0;i<(int)G[from].size();i++) {
      edge& e=G[from][i];
      if(e.to==to) {
     if(e.cap==0&&max_flow(from,to,1)==0) {
       max_flow(from,s,1);
       max_flow(t,to,1);
       return 1;
     }
      }
    }
    return 0;
  }
};
```

```
struct Fordfulkerson{
  const int INF = 1 << 28;
  struct edge{
    int to,cap,rev;
    edge(){}
    edge(int to,int cap,int
rev):to(to),cap(cap),rev(rev){}
  };

  vector<vector<edge> > G;
  vector<int> used;

  Fordfulkerson(){}
  Fordfulkerson(int V){init(V);}

  void init(int V){
    for(int i=0;i<(int)G.size();i++) G[i].clear();
    G.clear();
    used.clear();
    G.resize(V);
    used.resize(V);
  }

  void add_edge(int from,int to,int cap){
    G[from].push_back(edge(to,cap,G[to].size()));
    // undirected
    //
G[to].push_back(edge(from,cap,G[from].size()-1));
    // directed
    G[to].push_back(edge(from,0,G[from].size()-1));
  }

  int dfs(int v,int t,int f){
    if(v==t) return f;
    used[v]=true;
    for(int i=0;i<(int)G[v].size();i++){
      edge &e = G[v][i];
      if(!used[e.to] && e.cap > 0 ){
     int d=dfs(e.to,t,min(f,e.cap));
     if(d>0){
       e.cap-=d;
       G[e.to][e.rev].cap+=d;
       return d;
     }
      }
    }
    return 0;
  }

  int max_flow(int s,int t){
    int flow=0;
    for(;;){
      fill(used.begin(),used.end(),0);
      int f=dfs(s,t,INF);
      if(f==0) return flow;
      flow+=f;
    }
  }
};

struct PrimalDual{
  const int INF = 1<<28;
  typedef pair<int,int> P;
  struct edge{
    int to,cap,cost,rev;
    edge(){}
    edge(int to,int cap,int cost,int
rev):to(to),cap(cap),cost(cost),rev(rev){}
  };

  int V;
  vector<vector<edge> > G;
  vector<int> h,dist,prevv,preve;
  PrimalDual(){}
  PrimalDual(int V):V(V){init();}

  void init(){
    for(int i=0;i<(int)G.size();i++) G[i].clear();
    G.clear();
    h.clear();
    dist.clear();
    prevv.clear();
    preve.clear();
    G.resize(V);
    h.resize(V);
    dist.resize(V);
    prevv.resize(V);
    preve.resize(V);
  }

  void add_edge(int from,int to,int cap,int cost){

G[from].push_back(edge(to,cap,cost,G[to].size()));
    G[to].push_back(edge(from,0,-
cost,G[from].size()-1));
  }

  int min_cost_flow(int s,int t,int f){
    int res=0;
    fill(h.begin(),h.begin()+V,0);
    while(f>0){
      priority_queue<P,vector<P>,greater<P> > que;
      fill(dist.begin(),dist.begin()+V,INF);
      dist[s]=0;
      que.push(P(0,s));
      while(!que.empty()){
     P p=que.top();que.pop();
     int v=p.second;
     if(dist[v]<p.first) continue;
     for(int i=0;i<(int)G[v].size();i++){
       edge &e=G[v][i];
       if(e.cap>0&&dist[e.to]>dist[v]+e.cost+h[v]-
h[e.to]){
         dist[e.to]=dist[v]+e.cost+h[v]-h[e.to];
         prevv[e.to]=v;
         preve[e.to]=i;
         que.push(P(dist[e.to],e.to));
       }
     }
      }
      if(dist[t]==INF){
     return -1;
      }
      for(int v=0;v<V;v++) h[v]+=dist[v];

      int d=f;
      for(int v=t;v!=s;v=prevv[v]){
     d=min(d,G[prevv[v]][preve[v]].cap);
      }
      f-=d;
      res+=d*h[t];
      for(int v=t;v!=s;v=prevv[v]){
     edge &e=G[prevv[v]][preve[v]];
     e.cap-=d;
     G[v][e.rev].cap+=d;
      }
    }
    return res;
  }
};
```

動的計画法

```
int lcs(string X,string Y){
  const int N=max(X.size(),Y.size())+1;
  vector<vector<int> > c(N,vector<int>(N,0));
  int m = X.size();
  int n = Y.size();
  int maxl = 0;
  X = ' ' + X;
  Y = ' ' + Y;
  for(int i=1;i<=m;i++){
    for(int j=1;j<=n;j++){
      if(X[i]==Y[j]) c[i][j]=c[i-1][j-1]+1;
      else c[i][j]=max(c[i-1][j],c[i][j-1]);
      maxl=max(maxl,c[i][j]);
    }
  }
  return maxl;
}

struct LongestCommonSubstring{
  struct node{
    int value;
    node *next;
    node(int value,node
*next):value(value),next(next){}
  };
  const int INF=1LL<<55LL;
  string x,y;
  LongestCommonSubstring(){}
  LongestCommonSubstring(string x,string
y):x(x),y(y){}
  string lcs(){
    int n=x.size(),m=y.size();
    map<char,vector<int> > M;
    for(int j=m-1;j>=0;j--) M[y[j]].push_back(j);
    vector<int> xs(n+1,INF);
    xs[0]=-INF;
    vector<node* > link(n+1);
    for(int i=0;i<n;i++){
      if(M.count(x[i])){
     vector<int> ys=M[x[i]];
     for(int j=0;j<(int)ys.size();j++){
       int
k=distance(xs.begin(),lower_bound(xs.begin(),xs.end(
),ys[j]));
       xs[k]=ys[j];
       link[k]=new node(y[ys[j]],link[k-1]);
     }
      }
    }
    string res;
    int
l=distance(xs.begin(),lower_bound(xs.begin(),xs.end(
),INF-1))-1;
    for(node *p=link[l];p!=NULL;p=p->next)
      res.push_back(p->value);
    return res;
  }
};

int lis(int n,vector<int>& A){
  vector<int> L(A.size()+1,0);
  L[0]=A[0];
  int length=1;
  for(int i=1;i<n;i++){
    if(L[length-1]<A[i]){
      L[length++]=A[i];
    }else{
      *lower_bound(L.begin(),L.begin()
+length,A[i])=A[i];
    }
  }
  return length;
}

#define N 100
#define INF 1<<25
int n,p[N+1],m[N+1][N+1];
int matrixchainmultiplication(){
  for(int i=1;i<=n;i++) m[i][i]=0;
  for(int l=2;l<=n;l++){
    for(int i=1;i<=n-l+1;i++){
      int j=i+l-1;
      m[i][j]=INF;
      for(int k=i;k<=j-1;k++){
     m[i][j]=min(m[i][j],m[i][k]+m[k+1][j]
+p[i-1]*p[k]*p[j]);
      }
    }
  }
  return m[1][n];
}
```

データ構造

```
struct UnionFind{
  vector<int> r,p;
  UnionFind(){}
  UnionFind(int size){init(size);}
  void init(int size){
    r.resize(size,0);
    p.resize(size,0);
    for(int i=0;i<size;i++) r[i]=1,p[i]=i;
  }
  int find(int x){
    return (x==p[x]?x:p[x]=find(p[x]));
  }
  bool same(int x,int y){
    return find(x)==find(y);
  }
  void unite(int x,int y){
    x=find(x);y=find(y);
    if(x==y) return;
    if(r[x]<r[y]) swap(x,y);
    r[x]+=r[y];
    p[y]=x;
  }
};


struct WeightedUnionFind{
  vector<int> r,p,ws;
  WeightedUnionFind(){}
  WeightedUnionFind(int size){init(size);}
  void init(int size){
    r.resize(size,0);
    p.resize(size,0);
    ws.resize(size,0);
    for(int i=0;i<size;i++) r[i]=1,p[i]=i;
  }
  int find(int x){
    if(x==p[x]){
      return x;
    }else{
      int t=find(p[x]);
      ws[x]+=ws[p[x]];
      return p[x]=t;
    }
  }
  int weight(int x){
    find(x);
    return ws[x];
  }
  bool same(int x,int y){
    return find(x)==find(y);
  }
  void unite(int x,int y,int w){
    w+=weight(x);
    w-=weight(y);
    x=find(x);y=find(y);
    if(x==y) return;
    if(r[x]<r[y]) swap(x,y),w=-w;
    r[x]+=r[y];
    p[y]=x;
    ws[y]=w;
  }
  int diff(int x,int y){
    return weight(x)-weight(y);
  }
};


struct QuickFind{
  vector<int> r,p;
  vector<vector<int> > v;
  QuickFind(){}
  QuickFind(int size){init(size);}
  void init(int size){
    r.resize(size,0);
    p.resize(size,0);
    v.resize(size);
    for(int i=0;i<size;i++){
      r[i]=1,p[i]=i;
      v[i].resize(1,i);
    }
  }
  bool same(int x,int y){
    return p[x]==p[y];
  }
  void unite(int x,int y){
    x=p[x];y=p[y];
    if(x==y) return;
    if(r[x]<r[y]) swap(x,y);
    r[x]+=r[y];
    for(int i=0;i<(int)v[y].size();i++){
      p[v[y][i]]=x;
      v[x].push_back(v[y][i]);
    }
    v[y].clear();
  }
};

struct RollingHash{
  typedef unsigned long long ull;
  string S;
  ull B;
  vector<ull> hash,p;
  int len;
  RollingHash(){}
  RollingHash(string S,ull B=1000000007LL):S(S),B(B)
{init();};
  void init(){
    len=S.length();
    hash.resize(len+1);
    p.resize(len+1);
    hash[0]=0;p[0]=1;
    for(int i=0;i<len;i++){
      hash[i+1]=hash[i]*B+S[i];
      p[i+1]=p[i]*B;
    }
  }
  //S[l,r)
  ull find(int l,int r){
    return hash[r]-hash[l]*p[r-l];
  }
};
```

```
struct RollingHash2D{
  typedef unsigned long long ull;
  struct RollingHash{
    string S;
    ull B;
    vector<ull> hash,p;
    int len;
    RollingHash(){}
    RollingHash(string S,ull
B=1000000007LL):S(S),B(B){init();};
    void init(){
      len=S.length();
      hash.resize(len+1);
      p.resize(len+1);
      hash[0]=0;p[0]=1;
      for(int i=0;i<len;i++){
     hash[i+1]=hash[i]*B+S[i];
     p[i+1]=p[i]*B;
      }
    }
  //S[l,r)
    ull find(int l,int r){
      return hash[r]-hash[l]*p[r-l];
    }
  };
  vector<string> S;
  vector<RollingHash> rh;
  vector<vector<ull> > hash;
  vector<ull> p;
  int h,w,r,c;
  ull B;
  RollingHash2D(){}
  RollingHash2D(vector<string> S,int r,int c,ull
B=1000000009LL):S(S),r(r),c(c),B(B){init();};
  void init(){
    h=S.size();
    w=S[0].size();
    hash.resize(h+1,vector<ull>(w-c+1,0));
    rh.resize(h);
    for(int i=0;i<h;i++) rh[i]=RollingHash(S[i]);
    p.resize(h+1);
    p[0]=1;
    for(int i=0;i<h;i++) p[i+1]=p[i]*B;
    for(int j=0;j<w-c+1;j++){
      hash[0][j]=0;
      for(int i=0;i<h;i++)
     hash[i+1][j]=hash[i][j]*B+rh[i].find(j,j+c);
    }
  }
  //[(i,j),(i+r,j+c)]
  ull find(int i,int j){
    return hash[i+r][j]-hash[i][j]*p[r];
  }
};

struct SuffixArray{
  int n,k;
  string S;
  vector<int> r,r2,t,sa,lcp;
  SuffixArray(){}
  SuffixArray(string S):S(S){init();}
  void init(){
    n=S.size();
    r.resize(n+1,0);
    r2.resize(n+1,0);
    t.resize(n+1,0);
    sa.resize(n+1,0);
    lcp.resize(n+1,0);
    construct_sa();
    construct_lcp();
    construct_rmq();
  }
  bool compare_sa(int i,int j){
```

```
  if(r[i]!=r[j]) return r[i]<r[j];
  else{
   int ri=i+k<=n?r[i+k]:-1;
   int rj=j+k<=n?r[j+k]:-1;
   return ri<rj;
  }
 }
 void construct_sa(){
  n=S.length();
  for(int i=0;i<=n;i++){
   sa[i]=i;
   r[i]=i<n?S[i]:-1;
  }
  for(k=1;k<=n;k*=2){
   sort(sa.begin(),sa.end(),[&](const int &i,
const int &j){
    if(r[i]!=r[j]) return r[i]<r[j];
    else{
     int ri=i+k<=n?r[i+k]:-1;
     int rj=j+k<=n?r[j+k]:-1;
     return ri<rj;
    }
   });
   t[sa[0]]=0;
   for(int i=1;i<=n;i++){
   t[sa[i]]=t[sa[i-1]]+
(compare_sa(sa[i-1],sa[i])?1:0);
   }
   for(int i=0;i<=n;i++){
   r[i]=t[i];
   }
  }
 }
 bool contains(string T){
  int a=0,b=S.length()+1;
  while(a+1<b){
   int c=(a+b)/2;
```

```
   if(S.compare(sa[c],T.length(),T)<0) a=c;
   else b=c;
  }
  if(b==(int)S.length()+1) b--;
  return S.compare(sa[b],T.length(),T)==0;
 }

 // O(|T|*log|S|)
 int count(string T){
  int sl=S.length(),tl=T.length();
  int a[2],b[2];
  for(int i=0;i<2;i++){
   a[i]=0;
   b[i]=sl;
   while(a[i]+1<b[i]){
  int c=(a[i]+b[i])/2;
  if(S.compare(sa[c],tl,T)<0||
    (i&&S.compare(sa[c],tl,T)==0)) a[i]=c;
  else b[i]=c;
   }
  }
  if(S.compare(sa[b[0]],tl,T)!=0) return 0;
  if(a[1]<sl&&S.compare(sa[a[1]+1],tl,T)==0) a[1]+
+;
  if(b[0]> 0&&S.compare(sa[b[0]-1],tl,T)==0)
b[0]--;
  return a[1]-b[0]+1;
 }

 void construct_lcp(){
  for(int i=0;i<=n;i++) r2[sa[i]]=i;
  int h=0;
  lcp[0]=0;
  for(int i=0;i<n;i++){
   int j=sa[r2[i]-1];
   if(h>0) h--;
   for(;j+h<n&&i+h<n;h++){
     if(S[j+h]!=S[i+h]) break;
      }
      lcp[r2[i]-1]=h;
    }
  }

  int getlcp(int p,string &T,int d){
    int i=0;
    int len=min((int)T.length()-d,(int)S.length()-p-
d);
    while(i<len&&S[p+d+i]==T[d+i]) i++;
    return i;
  }

  struct RMQ{
    int n;
    vector<int> dat;
    const int def=INT_MAX;
    RMQ(){}
    RMQ(int n_){init(n_);}
    RMQ(int n_,vector<int>& a)
{init(n_);construct(n_,a);}
    void init(int n_){
      n=1;
      while(n<n_) n*=2;
      dat.clear();
      dat.resize(2*n-1,def);
    }
    void construct(int n_, vector<int>& a){
      for(int i=0;i<n_;i++) dat[i+n-1]=a[i];
      for(int i=n-2;i>=0;i--)
     dat[i]=min(dat[i*2+1],dat[i*2+2]);
    }
    void update(int k,int a){
      k+=n-1;
      dat[k]=a;
      while(k>0){
```

```
   k=(k-1)/2;
   dat[k]=min(dat[k*2+1],dat[k*2+2]);
    }
  }
  int query(int a,int b,int k,int l,int r){
    if(r<=a||b<=l) return def;
    if(a<=l&&r<=b) return dat[k];
    else{
   int vl=query(a,b,k*2+1,l,(l+r)/2);
   int vr=query(a,b,k*2+2,(l+r)/2,r);
   return min(vl,vr);
    }
  }
  int query(int a,int b){
    return query(a,b,0,0,n);
  }
};

RMQ rmq;
void construct_rmq(){
  rmq.init(n);
  rmq.construct(n,lcp);
}

// O(|T|+log|S|)
int count2(string T){
  int a[2],b[2];
  int sl=S.length(),tl=T.length();
  for(int i=0;i<2;i++){
    int p,l,r;
    p=tl;
    a[i]=0;
    b[i]=sl;
    l=getlcp(sa[a[i]],T,0);
    r=getlcp(sa[b[i]],T,0);
    while(a[i]+1<b[i]){
   int c=(a[i]+b[i])/2;
```

```
 //cout<<a[i]<<" "<<b[i]<<" "<<c<<endl;
 if(l>=r){
   int m=rmq.query(a[i],c);
   if(m<l) b[i]=c,r=m;
   else{
     int k=l+getlcp(sa[c],T,l);
     if(i){
       if(k==p||S[sa[c]+k]<T[k]) a[i]=c,l=k;
       else b[i]=c,r=k;
     }else{
       if(k==p) b[i]=c,r=k;
       else if(S[sa[c]+k]<T[k]) a[i]=c,l=k;
       else b[i]=c,r=k;
     }
   }
 }else{
   int m=rmq.query(c,b[i]);
   if(m<r) a[i]=c,l=m;
   else{
     int k=r+getlcp(sa[c],T,r);
     if(i){
       if(k==p||S[sa[c]+k]<T[k]) a[i]=c,l=k;
       else b[i]=c,r=k;
     }else{
       if(k==p) b[i]=c,r=k;
       else if(S[sa[c]+k]<T[k]) a[i]=c,l=k;
       else b[i]=c,r=k;
     }
   }
 }
  }
}

if(a[1]<sl&&getlcp(sa[a[1]+1],T,0)==tl) a[1]++;
if(b[0]> 0&&getlcp(sa[b[0]-1],T,0)==tl) b[0]--;

if(getlcp(sa[b[0]],T,0)!=tl) return 0;
```

```
    return a[1]-b[0]+1;
  }
};
```

その他

```
#define MOD 1000000007
#define MAX_N 100000
#define MAX_P 200005
int fact[MAX_P];
int extgcd(int a,int b,int& x,int& y){
  int d=a;
  if(b!=0){
    d=extgcd(b,a%b,y,x);
    y-=(a/b)*x;
  }else{
    x=1;y=0;
  }
  return d;
}
int mod_inverse(int a,int m){
  int x,y;
  extgcd(a,m,x,y);
  return (m+x%m)%m;
}

int euler_phi(int n){
  int res=n;
  for(int i=2;i*i<=n;i++){
    if(n%i==0){
      res=res/i*(i-1);
      for(;n%i==0;n/=i);
    }
  }
  if(n!=1) res=res/n*(n-1);
  return res;
}

int euler[MAX_N];

void euler_phi2(){
  for(int i=0;i<MAX_N;i++) euler[i]=i;
  for(int i=2;i<MAX_N;i++){
    if(euler[i]==i){
      for(int j=i;j<MAX_N;j+=i) euler[j]=euler[j]/
i*(i-1);
    }
  }
}

int mod_pow(int x,int n,int mod){
  int res=1;
  while(n>0){
    if(n&1) (res*=x)%=mod;
    (x*=x)%=mod;
    n>>=1;
  }
  return res;
}

void init(int p){
  fact[0]=1;
  for(int i=1;i<MAX_P;i++) fact[i]=(fact[i-1]*i)%p;
}

int mod_fact(int n,int p,int& e){
  e=0;
  if(n==0) return 1;
  int res=mod_fact(n/p,p,e);
  e+=n/p;
  if(n/p%2!=0)return res*(p-fact[n%p]) %p;
  return res*fact[n%p]%p;
}
int mod_comb(int n,int k,int p){
  if(n==k||k==0) return 1;
  int e1,e2,e3;
  int
a1=mod_fact(n,p,e1),a2=mod_fact(k,p,e2),a3=mod_fact(
n-k,p,e3);
  if(e1>e2+e3) return 0;
  return a1*mod_inverse(a2*a3%p,p)%p;
}

int expression(string,int&);
int term(string,int&);
int factor(string,int&);
int number(string,int&);

bool f;

int expression(string s,int& p){
  int res=term(s,p);
  while(p<(int)s.size()){
    if(s[p]=='+'){
      p++;
      res+=term(s,p);
      continue;
    }
    if(s[p]=='-'){
      p++;
      res-=term(s,p);
      continue;
    }
    break;
  }
  return res;
}
```

```
int term(string s,int& p){
  int res=factor(s,p);
  while(p<(int)s.size()){
    if(s[p]=='*'){
      p++;
      res*=factor(s,p);
      continue;
    }
    if(s[p]=='/'){
      p++;
      int tmp=factor(s,p);
      if(tmp==0){
      f=1;
      break;
      }
      res/=tmp;
      continue;
    }
    break;
  }
  return res;
}

int factor(string s,int& p){
  int res;
  if(s[p]=='('){
    p++;
    res=expression(s,p);
    p++;
  }else{
    res=number(s,p);
  }
  return res;
}

struct Dice{
  int s[6];
```

```
void roll(char c){
  //the view from above
  // N
  //W E
  // S
  //s[0]:top
  //s[1]:south
  //s[2]:east
  //s[3]:west
  //s[4]:north
  //s[5]:bottom
  int b;
  if(c=='E'){
    b=s[0];
    s[0]=s[3];
    s[3]=s[5];
    s[5]=s[2];
    s[2]=b;
  }
  if(c=='W'){
    b=s[0];
    s[0]=s[2];
    s[2]=s[5];
    s[5]=s[3];
    s[3]=b;
  }
  if(c=='N'){
    b=s[0];
    s[0]=s[1];
    s[1]=s[5];
    s[5]=s[4];
    s[4]=b;
  }
  if(c=='S'){
    b=s[0];
    s[0]=s[4];
    s[4]=s[5];
```

```
      s[5]=s[1];
      s[1]=b;
    }

    // migi neji (not verified)
    if(c=='R'){
      b=s[1];
      s[1]=s[3];
      s[3]=s[4];
      s[4]=s[2];
      s[2]=b;
    }
    if(c=='L'){
      b=s[1];
      s[1]=s[2];
      s[2]=s[4];
      s[4]=s[3];
      s[3]=b;
    }

  }
  int top() {
    return s[0];
  }
  int hash(){
    int res=0;
    for(int i=0;i<6;i++) res=res*256+s[i];
    return res;
  }
};
vector<Dice> makeDices(Dice d){
  vector<Dice> res;
  for(int i=0;i<6;i++){
    Dice t(d);
    if(i==1) t.roll('N');
    if(i==2) t.roll('S');
    if(i==3) t.roll('S'),t.roll('S');
```

```
    if(i==4) t.roll('L');
    if(i==5) t.roll('R');
    for(int k=0;k<4;k++){
      res.push_back(t);
      t.roll('E');
    }
  }
  return res;
}

int MOD=1000000009LL; //<- alert!!!
typedef vector<int> arr;
typedef vector<arr> mat;
inline arr mul(mat a,arr& b,int mod){
  arr res(b.size(),0);
  for(int i=0;i<(int)b.size();i++)
    for(int j=0;j<(int)a[i].size();j++)
      (res[i]+=a[i][j]*b[j])%=mod;
  return res;
}
inline mat mul(mat& a,mat& b,int mod){
  mat res(a.size(),arr(b[0].size(),0));
  for(int i=0;i<(int)a.size();i++)
    for(int j=0;j<(int)b[0].size();j++)
      for(int k=0;k<(int)b.size();k++)
     (res[i][j]+=a[i][k]*b[k][j])%=mod;
  return res;
}
mat base;
inline mat mat_pow(mat a,int n,int mod){
  if(base.empty()){
    base=mat(a);
    for(int i=0;i<(int)a.size();i++)
      for(int j=0;j<(int)a[i].size();j++)
     base[i][j]=(i==j);
  }
  mat res(base);
  while(n){
    if(n&1) res=mul(a,res,mod);
    a=mul(a,a,mod);
    n>>=1;
  }
  return res;
}
mat memo[100];
void init(mat a,int mod){
  base=mat(a);
  for(int i=0;i<(int)base.size();i++)
    for(int j=0;j<(int)base.size();j++)
      base[i][j]=i==j;
  memo[0]=a;
  for(int k=1;k<70;k++)
    memo[k]=mul(memo[k-1],memo[k-1],mod);
}
inline mat mat_pow2(int w,int n,int mod){
  mat res(base);
  int k=0;
  while(n){
    if(n&1) res=mul(memo[k],res,mod);
    n>>=1;
    k++;
  }
  return res;
}

struct frac{
  int num,dom;
  frac(){}
  frac(int num,int dom):num(num),dom(dom){}
  frac norm(){
    if(num==0) return frac(0,1);
    int tmp=__gcd(num,dom);
    return frac(num/tmp,dom/tmp);
  }
```

```
  frac norm2(){
    if(num==0) return frac(0,1);
    while(num<0) num+=dom;
    while(num>=dom) num-=dom;
    int tmp=__gcd(num,dom);
    return frac(num/tmp,dom/tmp);
  }
  frac operator+(frac a){return
frac(num*a.dom+a.num*dom,dom*a.dom).norm();}
  frac operator-(frac a){return frac(num*a.dom-
a.num*dom,dom*a.dom).norm();}
  frac operator*(frac a){return
frac(num*a.num,dom*a.dom).norm();}
  frac operator/(frac a){return
frac(num*a.dom,dom*a.num).norm();}
  frac operator*(int k){return
frac(num*k,dom).norm();}
  frac operator/(int k){return
frac(num,dom*k).norm();}
  bool operator<(const frac a)const{
    return num*a.dom<a.num*dom;
  }
  bool operator>(const frac a)const{
    return num*a.dom>a.num*dom;
  }
  bool operator==(const frac a)const{
    return num*a.dom==a.num*dom;
  }
  bool operator!=(const frac a)const{
    return num*a.dom!=a.num*dom;
  }
  bool operator<=(const frac a)const{
    return num*a.dom<=a.num*dom;
  }
  bool operator>=(const frac a)const{
    return num*a.dom>=a.num*dom;
  } };
```

```
const double EPS=1E-8;
typedef vector<double> vec;
typedef vector<vec> mat;
vec gauss_jordan(const mat& A,const vec& b){
  int n=A.size();
  mat B(n,vec(n+1));
  for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
      B[i][j]=A[i][j];
  for(int i=0;i<n;i++) B[i][n]=b[i];
  for(int i=0;i<n;i++){
    int pivot=i;
    for(int j=0;j<n;j++)
      if(abs(B[j][i])>abs(B[pivot][i])) pivot=j;
    swap(B[i],B[pivot]);
    if(abs(B[i][i])<EPS) return vec();
    for(int j=i+1;j<=n;j++) B[i][j]/=B[i][i];
    for(int j=0;j<n;j++){
      if(i!=j){
     for(int k=i+1;k<=n;k++) B[j][k]-=B[j][i]*B[i]
[k];
      }
    }
  }
  vec x(n);
  for(int i=0;i<n;i++) x[i]=B[i][n];
  return x;
}

typedef vector<double> arr;
typedef vector<arr> mat;

double det(mat A){
  int n=A.size();
  double res=1;
  for(int i=0;i<n;i++){
    int pivot=i;
    for(int j=i+1;j<n;j++)
      if(abs(A[j][i])>abs(A[pivot][i])) pivot=j;
    swap(A[pivot],A[i]);
    res*=A[i][i]*(i!=pivot?-1:1);
    if(abs(A[i][i])<EPS) break;
    for(int j=i+1;j<n;j++)
      for(int k=n-1;k>=i;k--)
     A[j][k]-=A[i][k]*A[j][i]/A[i][i];
  }
  return res;
}
bool isTriangle(double a1,double a2,double a3){
  if(a1+a2<=a3||a2+a3<=a1||a3+a1<=a2) return 0;
  return 1;
}
double tetrahedra(double OA,double OB,double
OC,double AB,double AC,double BC){
  if(!isTriangle(OA,OB,AB)) return 0;
  if(!isTriangle(OB,OC,BC)) return 0;
  if(!isTriangle(OC,OA,AC)) return 0;
  if(!isTriangle(AB,AC,BC)) return 0;
  mat A(5,arr(5,0));
  A[0][0]=    0;A[0][1]=AB*AB;
  A[0][2]=AC*AC;A[0][3]=OA*OA;A[0][4]=1;
  A[1][0]=AB*AB;A[1][1]=    0;A[1][2]=BC*BC;A[1]
[3]=OB*OB;A[1][4]=1;
  A[2][0]=AC*AC;A[2][1]=BC*BC;A[2][2]=    0;A[2]
[3]=OC*OC;A[2][4]=1;
  A[3][0]=OA*OA;A[3][1]=OB*OB;A[3][2]=OC*OC;A[3][3]=
0;A[3][4]=1;
  A[4][0]=    1;A[4][1]=    1;A[4][2]=    1;A[4][3]=
1;A[4][4]=0;
  //cout<<"det(A):"<<det(A)<<endl;
  if(det(A)<=0) return 0;
  return sqrt(det(A)/288.0);
}
```