TooWeakTooSlow, IIIT-Hyderabad Team Notebook

April 11, 2018

Contents

1 Basics

1.1 DosAndDonts

- Focus on the Problem, Not on the Leaderboard
- Review the code before submit. 2 min review << 20 min penalty
- Watch out for overflow,out-of-bound, i/j errors.
- cmp(val,*it) : upper_bound and cmp(*it,val) : lower_bound
- Stay Calm. Good Luck :)

1.2 Template

```
//TooWeakTooSlow
#include<bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC target("avx2, sse, sse2, sse3, ssse3, sse4,
    popcnt, abm, mmx, avx, tune=native")

using namespace std;
using namespace __gnu_pbds;

typedef tree<int ,null_type,less<int>, rb_tree_tag,
       tree_order_statistics_node_update> ordered_set;
// order_of_key (val): returns the no. of values less than val
// find_by_order (k): returns the kth largest element.(0-based)

typedef pair<int,int> II;
typedef vector< II >    VII;
typedef vector<int>   VI;
typedef vector< VI >  VVI;
typedef long long int  LL;

#define PB push_back
#define MP make_pair
#define F first
#define S second
#define SZ(a) (int)(a.size())
#define ALL(a) a.begin(),a.end()
#define SET(a,b) memset(a,b,sizeof(a))
#define FOR(i, a, b) for (int i = (a); i < (b); ++i)
#define REP(i, n) FOR(i, 0, n)

#define si(n) scanf("%d",&n)
#define dout(n) printf("%d\n",n)
#define sll(n) scanf("%lld",&n)
#define lldout(n) printf("%lld\n",n)
#define fast_io ios_base::sync_with_stdio(false);cin.tie(NULL)

#define TRACE

#ifdef TRACE
#define trace(...) __f(#__VA_ARGS__, __VA_ARGS__)
template <typename Arg1>
void __f(const char* name, Arg1&& arg1){
       cerr << name << " : " << arg1 << std::endl;
}template <typename Arg1, typename... Args>
void __f(const char* names, Arg1&& arg1, Args&&... args){
       const char* comma = strchr(names + 1,
           ',');cerr.write(names, comma - names) << " : " <<
           arg1<<" | ";__f(comma+1, args...);
}
#else
#define trace(...)
#endif
```

1.3 compilerSettings

- alias g++='g++ -g -O2 -std=gnu++11 -Wall'

1.4 vimSettings

- set nu autoindent hlsearch scrolloff=5 laststatus=2
- syntax on
- filetype plugin indent on
- autocmd BufNewFile *.cpp r ~/template.cpp | 1d
- set backspace=indent,eol,start
- set tabstop=2 softtabstop=2 shiftwidth=2 expandtab

2 DS

2.1 LazySegtree

```
int ST[4*N],lazy[4*N],A[N];
#define lc (x<<1)
#define rc (x<<1)|1
void push(int x,int l,int r){
  ST[x]+=lazy[x]; //Operation of lazy
  if(l==r-1)lazy[x]=0;
  if(!lazy[x])return;
  lazy[lc]+=lazy[x];
  lazy[rc]+=lazy[x];
  lazy[x]=0;//Propagate Lazy
}void up(int x){//Operation of Segtree
  ST[x] = min(ST[lc],ST[rc]);
}void build(int l=0,int r=N,int x=1){
  lazy[x]=0;//clear lazy
  if(l==r-1)return void(ST[x]=A[l]);
  int m = (l+r)/2;
  build(l,m,lc);
  build(m,r,rc);up(x);
}void update(int L,int R,int add,int l=0,int r=N,int x=1){
  push(x,l,r);int m = (l+r)/2;
  if(l>=R || r<=L)return;
  if(l>=L && r<=R){
    lazy[x]+=add;//operation of lazy update
    return push(x,l,r);
  }update(L,R,add,l,m,lc);
  update(L,R,add,m,r,rc);up(x);
}int query(int L,int R,int l=0,int r=N,int x=1){
  push(x,l,r);int m = (l+r)/2;
  if(l>=R||r<=L)return INF;//nothing here
  if(l>=L && r<=R)return ST[x];
  int la = query(L,R,l,m,lc);
  int ra = query(L,R,m,r,rc);
  return min(la,ra);//operation of segtree
}
```

2.2 PersistentSegtree

```
int L[N*LOGN],R[N*LOGN],ST[N*LOGN],blen,root[N],A[N];
//sparse persistent-segtree. range sum, initially 0
int update(int pos,int add,int l,int r,int id){
  if(l>pos || r<=pos)return id;
  int ID = ++blen, m=l+(r-l)/2;
  if(l==r-1)return (ST[ID]=ST[id]+add,ID);
  L[ID]=update(pos,add,l,m,L[id]);
  R[ID]=update(pos,add,m,r,R[id]);
  return (ST[ID]=ST[L[ID]]+ST[R[ID]],ID);
}root[0]=++blen;
for(int i=1;i<=n;i++)
  root[i]=update(A[i],1,0,MX,root[i-1]);
```

2.3 TreapBst

```
struct node{int val,prior,size;node *l,*r;};
typedef node* pnode;int sz(pnode t){return t?t->size:0;}
void upd_sz(pnode t){if(t)t->size = sz(t->l)+1+sz(t->r);}
void split(pnode t,pnode &l,pnode &r,int key){if(!t)l=r=NULL;
  else if(t->val<=key)split(t->r,t->r,r,key),l=t;//key in l
  else split(t->l,l,t->l,key),r=t;upd_sz(t);
}void merge(pnode &t,pnode l,pnode r){if(!l || !r)t=l?l:r;
  else if(l->prior> r->prior)merge(l->r,l->r,r),t=l;
  else merge(r->l,l,r->l),t=r;upd_sz(t);
}void insert(pnode &t,pnode it){if(!t) t=it;
  else if(it->prior>t->prior)split(t,it->l,it->r,it->val),t=it;
  else insert(t->val<it->val?t->r:t->l,it);upd_sz(t);
}void erase(pnode &t,int key){if(!t)return;
  else if(t->val==key){pnode x=t;merge(t,t->l,t->r);free(x);}
  else erase(t->val<key?t->r:t->l,key);upd_sz(t);
}void unite (pnode &t,pnode l, pnode r){
  if(!l||!r)return void(t=l?l:r);pnode lt,rt;
  if(l->prior<r->prior)swap(l,r);split(r,lt,rt,l->val);
  unite(l->l,l->l,lt);unite(l->r,l->r,rt);t=l;upd_sz(t);
}pnode init(int val){pnode ret = (pnode)malloc(sizeof(node));
  ret->val=val;ret->size=1;ret->prior=rand();ret->l=ret->r=NULL;
  return ret;}insert(init(x),head);
```

2.4 TreapIntervalTree

```
struct node{int prior,size,val,sum,lazy;node *l,*r;};
typedef node* pnode;//array value,segtree info,lazy update
int sz(pnode t){return t?t->size:0;}
void upd_sz(pnode t){if(t)t->size=sz(t->l)+1+sz(t->r);}
void lazy(pnode t){if(!t || !t->lazy)return;
  t->val+=t->lazy;/*operation of lazy*/t->sum+=t->lazy*sz(t);
  if(t->l)t->l->lazy+=t->lazy;//propagate lazy
  if(t->r)t->r->lazy+=t->lazy;t->lazy=0;
}void reset(pnode t){if(t)t->sum = t->val;//already propagated
}void combine(pnode& t,pnode l,pnode r){
```

```
  if(!l || !r)return void(t = l?l:r);//combine segtree ranges
  t->sum = l->sum + r->sum;
}void operation(pnode t){//operation of segtree
  if(!t)return;reset(t);//node == single element of array
  lazy(t->l);lazy(t->r);//imp:propagate lazy before combining
  combine(t,t->l,t);combine(t,t,t->r);
}void split(pnode t,pnode &l,pnode &r,int pos,int add=0){
  if(!t)return void(l=r=NULL);lazy(t);int cpos=add+sz(t->l);
  if(cpos<=pos)//element at pos goes to "l"
    split(t->r,t->r,r,pos,cpos+1),l=t;
  else split(t->l,l,t->l,pos,add),r=t;upd_sz(t);operation(t);
}void merge(pnode &t,pnode l,pnode r){//result/left/right array
  lazy(l);lazy(r);if(!l || !r) t = l?l:r;
  else if(l->prior>r->prior)merge(l->r,l->r,r),t=l;
  else merge(r->l,l,r->l),t=r;upd_sz(t);operation(t);
}pnode init(int val){pnode ret=(pnode)malloc(sizeof(node));
  ret->prior=rand();ret->size=1;ret->val=val;
  ret->sum=val;ret->lazy=0;return ret;
}int range_query(pnode t,int l,int r){//[l,r]
  pnode L,mid,R;split(t,L,mid,l-1);
  split(mid,t,R,r-l);/*note: r-l!*/int ans = t->sum;
  merge(mid,L,t);merge(t,mid,R);return ans;
}void range_update(pnode t,int l,int r,int val){//[l,r]
  pnode L,mid,R;split(t,L,mid,l-1);
  split(mid,t,R,r-l);/*note: r-l!*/t->lazy+=val; //lazy_update
  merge(mid,L,t);merge(t,mid,R);}
```

3 Geometry

3.1 2DGeometryTemplate

```
#define CT double
const CT EPS =1e-12;
#define EQ(a, b) (fabs((a) - (b)) <= EPS)
#define LT(a, b) ((a) < (b) - EPS)     /* less than */
struct Point {
  CT x, y;
  Point() {}
  Point(CT x, CT y) : x(x), y(y) {}
  Point(const Point &p) : x(p.x), y(p.y){}
  Point operator+(const Point &p)const{return
     Point(x+p.x,y+p.y);}
  Point operator-(const Point &p)const{return
     Point(x-p.x,y-p.y);}
  Point operator * (double c) const{return Point(x*c,y*c);}
  Point operator / (double c) const{return Point(x/c,y/c);}
};CT dot(Point p, Point q){return p.x*q.x+p.y*q.y;}
double dist2(Point p, Point q){return dot(p-q,p-q);}
CT cross(Point p, Point q){return p.x*q.y-p.y*q.x;}
ostream &operator<<(ostream &os, const Point &p){
  return os << "(" << p.x << "," << p.y << ")";
}bool operator < (const Point& a, const Point& b){
  if(!EQ(a.x, b.x))return LT(a.x, b.x);
  else return LT(a.y, b.y);
}bool operator == (const Point& a, const Point& b) {
  return EQ(a.x, b.x) && EQ(a.y, b.y);
}// rotate a point CCW or CW around the origin
Point RotateCCW90(Point p){ return Point(-p.y,p.x); }
Point RotateCW90(Point p){ return Point(p.y,-p.x); }
Point RotateCCW(Point p, double t) {//t in radians.
  return Point(p.x*cos(t)-p.y*sin(t),p.x*sin(t)+p.y*cos(t));
}// project point c onto line through a and b assuming a != b
Point ProjectPointLine(Point a, Point b, Point c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}// project point c onto line segment through a and b
Point ProjectPointSegment(Point a, Point b, Point c){
  double r = dot(b-a,b-a); if (EQ(r, 0)) return a;
  r = dot(c-a, b-a)/r; if (r < 0) return a;
  if (r > 1) return b;return a + (b-a)*r;
}//return line perp. to line through a & b, passing through c
pair<Point,Point> Perpendicularline(Point a,Point b,Point c){
  c=ProjectPointLine(a,b,c);if(a==c) a=b;
  return MP(c,c+RotateCW90(a-c));
}// compute distance from c to segment between a and b
double DistancePointSegment(Point a, Point b, Point c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}// compute distance between point (x,y,z) & plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
    double a, double b, double c, double d){
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}// determine if lines (a, b) and (c, d) are || or collinear
bool LinesParallel(Point a, Point b, Point c, Point d) {
  return EQ(cross(b-a, c-d), 0);
}bool LinesCollinear(Point a, Point b, Point c, Point d) {
  return LinesParallel(a, b, c, d) && EQ(cross(a-b, a-c),0)
    && EQ(cross(c-d, c-a), 0);
}// determine if line segment from a to b intersects with
```

```
// line segment from c to d
bool SegmentsIntersect(Point a, Point b, Point c, Point d) {
  if(b==d||b==c||a==d||a==c)return false;
  if (LinesCollinear(a, b, c, d)) {
    if (EQ(dist2(a, c),0) || EQ(dist2(a, d),0) ||
        EQ(dist2(b, c),0) || EQ(dist2(b, d),0))return true;
    if (dot(c-a,c-b)>0 && dot(d-a,d-b)>0 && dot(c-b,d-b)>0)
      return false;
    return true;
  }if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}// compute intersection of line passing through a & b with
    line
// passing through c & d, assuming unique intersection exists;
// for segment intersection, check if segments intersect first.
Point ComputeLineIntersection(Point a,Point b,Point c,Point d){
  b=b-a; d=c-d; c=c-a;assert(dot(b,b)>EPS&&dot(d,d)>EPS);
  return a + b*cross(c, d)/cross(b, d);
}// compute center of circle given three points
Point ComputeCircleCenter(Point a, Point b, Point c) {
  b=(a+b)/2;c=(a+c)/2;return //next line.
    ComputeLineIntersection(b,b+RotateCW90(a-b),c,c+RotateCW90(a-c));
}// determine if point is in a possibly non-convex polygon
// returns 1 for strictly interior points, 0 for strictly
// exterior points, and 0 or 1 for the remaining points.
bool PointInPolygon(const vector<Point> &p, Point q) {
  bool c = 0;REP(i, SZ(p)){
    int j = (i+1)%SZ(p);
    if((p[i].y<=q.y&&q.y<p[j].y||p[j].y<=q.y&&q.y<p[i].y)&&
        q.x<p[i].x+(p[j].x-p[i].x)*(q.y-p[i].y)/(p[j].y-p[i].y))
      c = !c;
  }return c;
}// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<Point> &p, Point q){
  REP(i,SZ(p)){auto
      pp=ProjectPointSegment(p[i],p[(i+1)%SZ(p)],q);
    if(dist2(pp, q) < EPS)return true;}
  return false;
}// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<Point> CircleLineIntersection(Point a,Point b,Point
    c,double r){
  vector<Point> ret; b = b-a; a = a-c;
  double A = dot(b, b); double B = dot(a, b);
  double C = dot(a, a) - r*r; double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.PB(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)ret.PB(c+a+b*(-B-sqrt(D))/A);
  return ret;
}// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<Point> CircleCircleIntersection(Point a,Point b,double
    r,double R){
  vector<Point> ret; double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d); double y = sqrt(r*r-x*x);
  Point v = (b-a)/d; ret.PB(a+v*x + RotateCCW90(v)*y);
  if (y > 0) ret.PB(a+v*x - RotateCCW90(v)*y);
  return ret;
}// Computes the area/centroid of (possibly nonconvex) polygon,
// assuming that the coordinates are listed in a clockwise or
// anticlockwise fashion. Note: the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<Point> &p) {
  double area = 0;
  REP(i, SZ(p)){ int j = (i+1) % SZ(p);
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }return area / 2.0;
}double ComputeArea(const vector<Point> &p) {
  return fabs(ComputeSignedArea(p));
}Point ComputeCentroid(const vector<Point> &p) {
  Point c(0,0); double scale = 6.0 * ComputeSignedArea(p);
  REP(i, SZ(p)){ int j = (i+1) % SZ(p);
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }return c / scale;
}// tests if a given polygon (in CW/CCW order) is simple
bool IsSimple(const vector<Point> &p) {
  REP(i, SZ(p)) FOR(k, i + 1, SZ(p)){
    int j = (i+1) % SZ(p), l = (k+1) % SZ(p);
    if (i == l || j == k) continue;
    if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
      return false;
  }return true;}
#define sqr(x) (x)*(x)
#define REMOVE_REDUNDANT
```

```
CT area2(Point a,Point b,Point c){return cross(b-a,c-a);}//2*A
#ifdef REMOVE_REDUNDANT
bool between(Point &a,Point &b,Point &c){// b is between a & c
  return EQ(area2(a, b, c),0) && (a.x-b.x)*(c.x-b.x) <= 0
    && (a.y-b.y)*(c.y-b.y) <= 0;}
#endif
void ConvexHull(vector<Point> &pts) {
  sort(ALL(pts));pts.erase(unique(ALL(pts)), pts.end());
  vector<Point> up, dn;
  REP(i, SZ(pts)) {
    while(SZ(up)>1&&area2(up[SZ(up)-2],up.back(),pts[i])>=0)
      up.pop_back();
    while(SZ(dn)>1&&area2(dn[SZ(dn)-2],dn.back(),pts[i])<=0)
      dn.pop_back();
    up.PB(pts[i]);dn.PB(pts[i]);
  }pts = dn;
  for (int i = (int) SZ(up) - 2; i >= 1; i--) pts.PB(up[i]);
#ifdef REMOVE_REDUNDANT
  if (SZ(pts) <= 2) return;
  dn.clear();dn.PB(pts[0]);dn.PB(pts[1]);
  FOR(i, 2, SZ(pts)){
    if(between(dn[SZ(dn)-2],dn[SZ(dn)-1],pts[i]))dn.pop_back();
    dn.PB(pts[i]);}
  if (SZ(dn) >= 3 && between(dn.back(), dn[0], dn[1])) {
    dn[0] = dn.back();dn.pop_back();
  }pts = dn;
#endif}//returns (B-A)X(C-A)
#define Det(a,b,c) ((b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x))
bool in_convex(vector<Point>& l, Point p){
  int a = 1, b = SZ(l)-1, c;
  if (Det(l[0], l[a], l[b]) > 0) swap(a,b);
  // orientation of area, a is above 0 and b below 0
  // Allow on edge --> if (Det... > 0 || Det ... < 0)
  if(Det(l[0],l[a],p)>=0||Det(l[0],l[b],p)<=0)return false;
  while(abs(a-b) > 1) {c = (a+b)/2;
    if (Det(l[0], l[c], p) > 0)b = c; else a = c;
  }// Alow on edge --> return Det... <= 0
  return Det(l[a], l[b], p) <= 0;}
#define line pair<Point,Point>
#define NEED3RDTANGENT
// need to be careful when tangent has single common point with
// both circles specially when one cricle lies inside other
vector<line> find_tangent(Point a, Point b, CT r1, CT r2) {
  vector<line> Q;if(dist2(a, b) <= sqr(r1 - r2))return Q;
  int f = 0; if(r2 > r1) swap(a, b), swap(r1, r2), f = 1;
  if(abs(r2 - r1) <= EPS) {
    line m=Perpendicularline(a,b,a),n=Perpendicularline(a,b,b);
    vector<Point> l1 = CircleLineIntersection(m.F, m.S, a, r1),
      l2 = CircleLineIntersection(n.F, n.S, b, r2);
    assert(SZ(l1) == 2 && SZ(l2) == 2);
    if(cross(b-a,l1[0]-b)*cross(b-a,l2[0]-b)<0)swap(l2[0],l2[1]);
    Q.PB(MP(l1[0], l2[0]));Q.PB(MP(l1[1], l2[1]));
  } else {Point out = (b * r1 - a * r2) / (r1 - r2);
    assert(dist2(out, a) >= r1 && dist2(out, b) >= r2);
    vector<Point> l1 = CircleCircleIntersection(a, out, r1,
        sqrt(dist2(out, a) - sqr(r1))),
      l2 = CircleCircleIntersection(b, out, r2,
          sqrt(dist2(out, b) - sqr(r2)));
    assert(SZ(l1) == 2 && SZ(l2) == 2);
    if(cross(b-a,l1[0]-b)*cross(b-a,l2[0]-b)<0)swap(l2[0],l2[1]);
    Q.PB(MP(l1[0], l2[0]));Q.PB(MP(l1[1], l2[1]));
  }if (dist2(a, b) > sqr(r1 + r2) + EPS) {
    Point out = (b * r1 + a * r2) / (r1 + r2);
    assert(dist2(out, a) >= r1 && dist2(out, b) >= r2);
    vector<Point> l1 = CircleCircleIntersection(a, out, r1,
        sqrt(dist2(out, a) - sqr(r1))),
      l2 = CircleCircleIntersection(b, out, r2,
          sqrt(dist2(out, b) - sqr(r2)));
    assert(SZ(l1) == 2 && SZ(l2) == 2);
    if(cross(b-a,l1[0]-b)*cross(b-a,l2[0]-b)>0)swap(l2[0],
       l2[1]);
    Q.PB(MP(l1[0], l2[0])); Q.PB(MP(l1[1], l2[1]));
  } else if (abs(sqr(r1 + r2) - dist2(a, b)) < EPS) {
#ifdef NEED3RDTANGENT
    Point out = (b * r1 + a * r2) / (r1 + r2);
    Q.PB(Perpendicularline(a, b, out));
#endif}
    if (f == 1) {REP(i, Q.size()) swap(Q[i].F, Q[i].S);}
    return Q;}
```

3.2 3DGeometryTemplate

```
#define eq(a,b) (fabs((a) - (b)) < EPS)
class point{
  public: double x,y,z;
    point(){x=y=z=0;}
    point(double _x,double _y,double _z=0):x(_x),y(_y),z(_z){}
```

```
    point(point &p){x=p.x;y=p.y;z=p.z;}
    bool operator == (point p) const{return eq(x,p.x) &&
       eq(y,p.y) && eq(z,p.z);}
    bool operator < (point p) const{
      if(eq(x,p.x) && eq(y,p.y)) return z<p.z;
      if(eq(x,p.x))return y<p.y; return x<p.x;
    }point operator + (point p) const {return
       point(x+p.x,y+p.y,z+p.z);}
    point operator - () const {return
       point(x-p.x,y-p.y,z-p.z);}
}null;point tovect(point a,point b){return
    point(b.x-a.x,b.y-a.y,b.z-a.z);}
point cross(point a,point b){return point(a.y*b.z - a.z*b.y ,
    -(a.x*b.z-a.z*b.x) , a.x*b.y-b.x*a.y);
}double dot(point a,point b){return a.x*b.x + a.y*b.y +
    a.z*b.z;}
double scalarTripleProduct(point a,point b, point c){return
    dot(a,cross(b,c));}
double mod(point v){return sqrt(dot(v,v));}
point norm(point v){double d =
    mod(v);v.x/=d;v.y/=d;v.z/=d;return v;}
double angle(point a,point b){a = norm(a);b = norm(b);return
    acos(dot(a,b));}
//********** LINE ***************
class line{
  public: point a,b; line(){}
    line(point x ,point y):a(x),b(tovect(x,y)){}
};bool areParallel(line l1,line l2){
  return cross(l1.b,l2.b)==null &&
      !(cross(point(l1.a,l2.a),l2.b) == null);
}bool areSame(line l1,line l2){
  return cross(l1.b,l2.b)==null &&
      (cross(point(l1.a,l2.a),l2.b) == null);
}bool areIntersect(line l1,line l2){
  return !(cross(l1.b,l2.b)==null) &&
      (fabs(scalarTripleProduct(point(l1.a,l2.a),l1.b,l2.b))<EPS);
}bool areIntersect(line l1,line l2,point& p1){
  if(!(cross(l1.b,l2.b)==null) &&
      (fabs(scalarTripleProduct(point(l1.a,l2.a),l1.b,l2.b))<EPS)){
    point temp = cross(l2.b,l1.b);
    double k2 =
       dot(cross(point(l2.a,l1.a),l1.b),temp)/dot(temp,temp);
    p1 = point(l2.a.x + k2*l2.b.x , l2.a.y + k2*l2.b.y,l2.a.z
       + k2*l2.b.z);
    return true;}return false;
}bool areSkew(line l1,line l2){
  return !areParallel(l1,l2) && !areSame(l1,l2) &&
      !areIntersect(l1,l2);
}/*************** PLANE **************/
class plane{ //point and normal vector
  public: point a,n; plane(){}
    plane(point _x,point _a,point _b){
      n = cross(_a,_b);a = _x;
      if(n == null)n =
         cross(tovect(_x,point(_a.x,_a.y,_a.z)),_b);
    }plane(line l1,line l2){a = l1.a;n = cross(l1.b,l2.b);
      if(n == null)n = cross(tovect(l1.a,l2.a),l1.b);
    }plane(plane & p){a = p.a;n = p.n;}
    plane(const plane & p){a = p.a;n = p.n;}
    bool operator == (plane p) const{
      return cross(n,p.n)==vect(0,0,0) &&
         (fabs(dot(n,tovect(a,p.a)))<EPS);
    }bool operator < (plane p) const{
      if(cross(n,p.n)==null &&
         fabs(dot(n,tovect(a,p.a)))<EPS)return false;
      if(a==p.a)return n<p.n;return a < p.a;}
};/****************Miscellaneous**********************/
// distance from point (x, y, z) to plane aX + bY + cZ + d = 0
double ptPlaneDist(double x, double y, double z,
    double a, double b, double c, double d){
  return abs(a*x + b*y + c*z + d) / sqrt(a*a + b*b + c*c);
}// distance from point (px, py, pz) to line (x1, y1, z1)-(x2,
    y2, z2)
// (or ray, or segment; in the case of the ray, the endpoint
    is the first point)
const int LINE = 0, SEGMENT = 1, RAY = 2;
double ptLineDistSq(double x1, double y1, double z1,
    double x2, double y2, double z2, double px, double py,
       double pz,
    int type){double pd2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) +
       (z1-z2)*(z1-z2);
  double x,y,z; if(eq(pd2,0)){x = x1;y = y1;z = z1;}
  else{ double u = ((px-x1)*(x2-x1) + (py-y1)*(y2-y1) +
     (pz-z1)*(z2-z1)) / pd2;
```

```
    x = x1 + u * (x2 - x1);y = y1 + u * (y2 - y1);z = z1 + u *
        (z2 - z1);
    if(type!=LINE && u < 0){x = x1;y = y1;z = z1;}
    if(type==SEGMENT && u > 1.0){x = x2;y = y2;z = z2;}
  }return (x-px)*(x-px) + (y-py)*(y-py) + (z-pz)*(z-pz);
}double ptLineDist(double x1, double y1, double z1,
    double x2, double y2, double z2, double px, double py,
        double pz,
    int type) {return sqrt(ptLineDistSq(x1, y1, z1, x2, y2,
        z2, px, py, pz, type));
}//projection of point p on plane A
point getProjection(point p,plane A){
  double a = A.n.x, b = A.n.y, c = A.n.z;
  double d = A.a.x, e = A.a.y, f = A.a.z;
  double t = (a*d - a*p.x + b*e - b*p.y + c*f - c*p.z)/(a*a +
      b*b + c*c);
  return point(p.x + t*a,p.y + t*b, p.z + t*c);
}//check if point p is in triangle A,B,C - 3D
bool ok(double x){return x>=0 && x<=1.0;}
bool inTriangle(point p,point A,point B,point C){
  double Area = mod(cross(tovect(A,B),tovect(A,C)));
  double alpha = mod(cross(tovect(p,B),tovect(p,C)))/Area;
  double beta = mod(cross(tovect(p,C),tovect(p,A)))/Area;
  double gamma = 1 - alpha - beta;
  return ok(alpha) && ok(beta) && ok(gamma);
}//rotate point A about axis B-->C by theta. C should be unit
    vector along the axis
point rotate(point A,point B,point C,double theta){
  double x = A.x, y = A.y, z = A.z;
  double a = B.x, b = B.y, c = B.z;
  double u = C.x, v = C.y, w = C.z;
  point ret;
  ret.x = (a*(sq(v)+sq(w)) - u*(b*v + c*w - u*x - v*y - w*z))
      * (1 - cos(theta)) + x*cos(theta) + (-c*v + b*w - w*y +
      v*z)*sin(theta);
  ret.y = (b*(sq(u)+sq(w)) - v*(a*u + c*w - u*x - v*y - w*z))
      * (1 - cos(theta)) + y*cos(theta) + (+c*u - a*w + w*x -
      u*z)*sin(theta);
  ret.z = (c*(sq(v)+sq(u)) - w*(a*u + b*v - u*x - v*y - w*z))
      * (1 - cos(theta)) + z*cos(theta) + (-b*u + a*v - v*x +
      u*y)*sin(theta);
  return ret;}
```

4 Graphs

4.1 AuxillaryTree

```
//Q[]: array containing k nodes of auxillary tree
//arr[] : arrival time of nodes
//anc(p,u) : returns true if p is ancestor of u
//VI tree[N] : final auxillary tree with O(k) nodes
bool cmp(int u,int v){return arr[u]<arr[v];}
int create_tree(){//return root of tree
  set<int> S;//get distinct nodes
  REP(i,k)S.insert(Q[i]);k=0;for(auto it : S)Q[k++]=it;
  sort(Q,Q+k,cmp);int kk = k;//distinct initial nodes
  //add lca of adjacent pairs
  for(int i=0;i<kk-1;i++){int x = lca(Q[i],Q[i+1]);
    if(S.count(x))continue;Q[k++]=x;S.insert(x);
  }sort(Q,Q+k,cmp);stack<int> s;s.push(Q[0]);
  for(int i=1;i<k;i++){
    while(!anc(s.top(),Q[i]))s.pop();
    tree[s.top()].PB(Q[i]);tree[Q[i]].PB(s.top());
    s.push(Q[i]);}return Q[0];}
```

4.2 BCC

```
VI g[N],tree[N],st;bool isArtic[N];int U[M],V[M],low[N];
int ord[N],depth[N],col[N],C,T,compNo[N],extra[N];
//For all [1,n+C] whose extra[i]=0 is part of Block-Tree.
//1-Based.Everything from [1,C] : type B & [C,n+C] : type C.
void dfs(int i){//Doesnt work for multi-edges.Remove them
  low[i]=ord[i]=T++;for(int j=0;j<SZ(g[i]);j++){
    int ei=g[i][j],to = adj(i,ei);
    if(ord[to]==-1){
      depth[to]=depth[i]+1;st.PB(ei);dfs(to);
      low[i] = min(low[i],low[to]);
      if(ord[i]==0||low[to]>=ord[i]){
        if(ord[i]!=0||j>=1)isArtic[i] = true;
        ++C;
        while(!st.empty()){
          int fi=st.back();st.pop_back();col[fi]=C;
          if(fi==ei)break;
        }
      }
    }else if(depth[to]<depth[i]-1){
      low[i] = min(low[i],ord[to]);st.PB(ei);}}}
void run(int n){
```

```
  SET(low,-1);SET(depth,-1);
  SET(ord,-1);SET(col,-1);SET(isArtic,0);st.clear();C=0;
  for(int i=1;i<=n;++i)
    if(ord[i]==-1)
      T = 0,dfs(i);
}void buildTree(int n){
  run(n);SET(compNo,-1);VI tmpv;SET(extra,-1);
  for(int i=1;i<=n;i++){
    tmpv.clear();for(auto e:g[i])tmpv.PB(col[e]);
    sort(ALL(tmpv));tmpv.erase(unique(ALL(tmpv)),tmpv.end());
    //handle isolated vertics
    if(tmpv.empty()){compNo[i]=C+i;extra[C+i]=0;continue;}
    if(SZ(tmpv)==1){//completely in 1 comp.
      compNo[i] = tmpv[0];extra[tmpv[0]]=0;
    }else{ // its  an articulation vertex.
      compNo[i]=C+i;extra[C+i]=0;
      for(auto j:tmpv){
        extra[j]=0;tree[C+i].PB(j);tree[j].PB(C+i);
      }}}}
```

4.3 BellmanFord

```
void BellmanFord(int s){
  REP(i, n)d[i] = INF;d[s] = 0;
  REP(step, n + 1)REP(i, m){
    int from = U[i], to = V[i], wt = W[i];
    if(d[from] + wt < d[to]){
      if(step == n){
        return void(puts("Negative Cycle Found"));
      }d[to] = d[from] + wt;
    }//To solve differential constraints problem using BF,
  }//For each constraint X_i - X_j <= C_i, add an edge from
}//X_j -> X_i of wt C_i. Connect a source s to all vertices
//X_i and run BF. -ve cycle -> Not Possible, else d[i] forms
//a valid assignment. BF also minimizes max{X_i} - min{X_i}
//This works coz finally,d[i]<=d[j]+C_i for all constraints.
```

4.4 BridgeTree

```
VI tree[N],g[N];bool isbridge[M];
int U[M],V[M],vis[N],arr[N],T,dsu[N];
int f(int x){return dsu[x]=(dsu[x]==x?x:f(dsu[x]));}
void merge(int a,int b){dsu[f(a)]=f(b);}
int dfs0(int u,int edge){ //mark bridges
  vis[u]=1;arr[u]=T++;int dbe=arr[u];
  for(auto e : g[u]){int w = adj(u,e);
    if(!vis[w])dbe = min(dbe,dfs0(w,e));
    else if(e!=edge)dbe = min(dbe,arr[w]);
  }if(dbe==arr[u]&&edge!=-1)isbridge[edge]=true;
  else if(edge!=-1)merge(U[edge],V[edge]);
  return dbe;
}void buildBridgeTree(int n,int m){
  for(int i=1;i<=n;i++)dsu[i]=i;int x,y;
  for(int i=1;i<=n;i++)if(!vis[i])dfs0(i,-1);
  for(int i=1;i<=m;i++)if((x=f(U[i]))!=(y=f(V[i])))
      tree[x].PB(y),tree[y].PB(x);}
```

4.5 CentroidDecomposition

```
VI g[N];int sub[N],nn,U[N],V[N],done[N];
void dfs1(int u,int p){
  sub[u]=1;nn++;
  for(auto e:g[u]){
    int w = adj(u,e);
    if(w!=p && !done[e])
      dfs1(w,u),sub[u]+=sub[w];}}
int dfs2(int u,int p){
  for(auto e:g[u]){
    if(done[e])continue;
    int w = adj(u,e);
    if(w!=p && sub[w]>nn/2)
      return dfs2(w,u);
  }return u;}
void decompose(int root,int p){
  nn=0;dfs1(root,root);root=dfs2(root,root);
  if(p==-1)p=root;//fuck centroid :)
  for(auto e:g[root]){
    if(done[e])continue;
    done[e]=1;int w = adj(root,e);
    decompose(w,root);}}
```

4.6 Circulation

```
// Configure: MAXE (at least 2 * calls_to_edge)
// - init(n) --> AddEdge(x,y,c,w) --> run();
// - AddEdge(x, y, c, w) edge x->y with capacity c and cost w
namespace Circu {const int MAXV = 1000100, MAXE = 1000100;
  int how[MAXV],good[MAXV],bio[MAXV],cookie=1,to[MAXE];
  int from[MAXE],V,E;LL cap[MAXE],cost[MAXE],dist[MAXV];
  void init(int n){V=n;E=0;}
  void AddEdge(int x,int y,LL c, LL w) {
    from[E]=x;to[E]=y;cap[E]=c;cost[E]=+w;++E;
```

```
    from[E]=y;to[E]=x;cap[E]=0;cost[E]=-w;++E;
  }void reset(){REP(i, V) dist[i]=0,how[i]=-1;}
  bool relax(){bool ret = false;
    REP(e, E)if(cap[e]){ int x=from[e],y=to[e];
      if(dist[x]+cost[e]<dist[y]){
        dist[y]=dist[x]+cost[e];
        how[y]=e;ret=true;
      }}return ret;
  }LL cycle(int s,bool flip = false){
    int x=s;LL c=cap[how[x]],sum = 0;
    do{int e=how[x];c = min(c,cap[e]);x = from[e];
    }while (x!=s);
    do{int e = how[x];
      if(flip){cap[e]-=c;cap[e^1]+=c;
      }sum += cost[e]*c;x=from[e];
    }while(x!=s);
    return sum;
  }LL push(int x){
    for(++cookie; bio[x]!=cookie; x=from[how[x]]){
      if(!good[x]||how[x]==-1||cap[how[x]]==0)return 0;
      bio[x]=cookie;good[x]=false;
    }return cycle(x) >= 0 ? 0 : cycle(x, true);
  }LL run(){
    reset();LL ret = 0;
    REP(step,2*V){if(step==V)reset();if(!relax())continue;
      REP(i, V) good[i] = true;
      REP(i, V) if(LL w=push(i))ret+=w,step=0;
    }return ret;}}
```

4.7 DSU

```
int F(int x){//dsu maintaing 2-coloring
  if(x==dsu[x])return x;//of each tree
  int p = F(dsu[x]);//in the forest
  C[x] ^= C[dsu[x]];return dsu[x]=p;
}bool Union(int a,int b){
  int x=F(a),y=F(b);if(x==y)return 0;
  if(sz[x]>sz[y])swap(x,y),swap(a,b);
  int p = (C[a]==C[b]);C[x] ^= p;
  sz[y] += sz[x];dsu[x] = y;
  F(a);F(b);return true;}
```

4.8 Dinics

```
// Max flow of directed weighted graph from source to sink.
// init(n)-->AddEdge(x,y,c1,c2)-->run(src,sink).
// AddEdge(x,y,c1,c2)adds x->y of cap c1 and y->x of cap c2
namespace Dinic{// MAXE = 2*(# calls to AddEdge);
  const int MAXV=int(1e5)+10,MAXE=int(2e5)+10;
  const LL INF=1e18;LL cap[MAXE];int V,E,last[MAXV];
  int dist[MAXV],curr[MAXV],next[MAXE],adj[MAXE];
  void init(int n){V=n;E=0;REP(i,V)last[i]=-1;}
  void AddEdge(int x,int y,LL c1,LL c2){
    adj[E]=y;cap[E]=c1;next[E]=last[x];last[x]=E++;
    adj[E]=x;cap[E]=c2;next[E]=last[y];last[y]=E++;
  }LL push(int x,int sink,LL flow){
    if(x==sink)return flow;
    for(int &e=curr[x];e!=-1;e=next[e]){
      int y=adj[e];
      if(cap[e]&&dist[x]+1==dist[y])
        if(LL f=push(y,sink,min(flow,cap[e])))
          return cap[e]-=f,cap[e^1]+=f,f;
    }return 0;
  }LL run(int src,int sink){LL ret=0;
    while(1){
      REP(i,V)curr[i]=last[i],dist[i]=-1;
      queue<int> Q;Q.push(src),dist[src]=0;
      while(!Q.empty()){
        int x=Q.front();Q.pop();
        for(int e=last[x];e!=-1;e=next[e]){
          int y=adj[e];
          if(cap[e]&&dist[y]==-1)
            Q.push(y),dist[y]=dist[x]+1;
        }}if(dist[sink]==-1)break;
      while(LL f=push(src,sink,INF))ret+=f;
    }return ret;}}
```

4.9 DominatorTree

```
VI g[N],tree[N],rg[N],bucket[N];int sdom[N],par[N];
int dom[N],dsu[N],label[N],arr[N],rev[N],T;
int Find(int u,int x=0){//1-Based directed graph
  if(u==dsu[u])return x?-1:u;int v = Find(dsu[u],x+1);
  if(v<0)return u;if(sdom[label[dsu[u]]]<sdom[label[u]])
    label[u] = label[dsu[u]];dsu[u] = v;return x?v:label[u];
}void Union(int u,int v){dsu[v]=u;}//yup,its correct :)
void dfs0(int u){T++;arr[u]=T;rev[T]=u;label[T]=T;
  sdom[T]=T;dsu[T]=T;for(auto w : g[u]){
    if(!arr[w])dfs0(w),par[arr[w]]=arr[u];
    rg[arr[w]].PB(arr[u]);}}//Build Dominator tree(in main)
```

```
dfs0(1);n=T;for(int i=n;i>=1;i--){for(int j=0;j<SZ(rg[i]);j++)
    sdom[i] = min(sdom[i],sdom[Find(rg[i][j])]);
  if(i>1)bucket[sdom[i]].PB(i);for(auto w : bucket[i]){
    int v = Find(w);if(sdom[v]==sdom[w])dom[w]=sdom[w];
    else dom[w] = v;}if(i>1)Union(par[i],i);
}for(int i=2;i<=n;i++){if(dom[i]!=sdom[i])dom[i]=dom[dom[i]];
  tree[rev[i]].PB(rev[dom[i]]);tree[rev[dom[i]]].PB(rev[i]);}
```

4.10 EdmondBlossom

```
//- graph stored in adj : 0-based. O(n^3) per call
//- match[i] stores vertex matched to i. -1 if unmatched
vector<int> adj[MAXN];
int p[MAXN], base[MAXN], match[MAXN];
int lca(int nodes, int u, int v){
  vector<bool> used(nodes);
  for (;;) {u = base[u];used[u] = true;
    if (match[u] == -1) {break;}
    u = p[match[u]];}
  for (;;) {v = base[v];if (used[v]) {return v;}
    v = p[match[v]];}}
void mark_path(vector<bool> &blossom,int u,int b,int child){
  for (; base[u] != b; u = p[match[u]]) {
    blossom[base[u]] = true;
    blossom[base[match[u]]] = true;
    p[u] = child;child = match[u];}}
int find_path(int nodes, int root) {
  vector<bool> used(nodes);
  for (int i = 0; i < nodes; ++i){p[i] = -1;base[i] = i;}
  used[root] = true;queue<int> q;q.push(root);
  while (!q.empty()) { int u = q.front();q.pop();
    for (int j = 0; j < SZ(adj[u]); j++) {int v = adj[u][j];
      if (base[u] == base[v] || match[u] == v) {continue;}
      if (v == root || (match[v] != -1 && p[match[v]] != -1)) {
        int curr_base = lca(nodes, u, v);
        vector<bool> blossom(nodes);
        mark_path(blossom, u, curr_base, v);
        mark_path(blossom, v, curr_base, u);
        for (int i = 0; i < nodes; i++) {
          if (blossom[base[i]]) {base[i] = curr_base;
            if (!used[i]) {used[i] = true;q.push(i);}}}
      } else if (p[v] == -1) {p[v] = u;
        if (match[v] == -1) {return v;}
          v = match[v];used[v] = true;q.push(v);
      }}}return -1;}
int edmonds(int nodes) {
  for (int i = 0; i < nodes; i++) {match[i] = -1;}
  for (int i = 0; i < nodes; i++) {
    if (match[i] == -1) { int u, pu, ppu;
      for (u = find_path(nodes, i); u != -1; u = ppu) {
        pu = p[u]; ppu = match[pu];match[u] = pu;
        match[pu] = u;}}}int matches = 0;
  for (int i = 0; i < nodes; i++) {
    if (match[i] != -1) {matches++;
    }}return matches/2;}
```

4.11 HLD

```
int U[N],V[N],W[N],baseArray[N],DP[LOGN][N],level[N],sub[N];
int chainParent[N],chainHead[N],blen,chainNo[N],pos[N],nchain;
void HLD(int u,int ee){//edge list graph.graph is 1-based.
  baseArray[blen]=W[ee];pos[u]=blen;blen++;chainNo[u]=nchain;
  int sc=-1,mx=0;
  for(auto e : g[u]){
    if(e==ee)continue;
    int w = adj(u,e);
    if(sub[w]>mx)sc = e,mx = sub[w];
  }if(sc==-1)return;
  HLD(adj(u,sc),sc);
  for(auto e : g[u]){
    if(e==ee||e==sc)continue;
    int w = adj(u,e);nchain++;
    chainParent[nchain]=u;chainHead[nchain]=w;
    HLD(w,e);}}
void dfs(int u,int ee){sub[u]=1;
  for(auto e : g[u]){if(e==ee)continue;
    int w=adj(u,e);level[w]=level[u]+1;
    DP[0][w]=u;dfs(w,e);sub[u]+=sub[w];
  }}void preprocess(){DP[0][1]=1;dfs(1,0);
  chainHead[nchain]=chainParent[nchain]=1;HLD(1,0);}
```

4.12 HopcroftKarp

```
//init(n1,n2):takes no of vertices on left and right
//addEdge(u,v):node u on left and v on right(0-based)
const int MAXN1=50000,MAXN2=50000,MAXM=150000;
int n1,n2,edges,last[MAXN1],prv[MAXM],head[MAXM],Q[MAXN1];
int matching[MAXN2],dist[MAXN1];bool used[MAXN1],vis[MAXN1];
void init(int _n1,int
    _n2){n1=_n1;n2=_n2;edges=0;fill(last,last+n1,-1);}
```

```
void addEdge(int u,int
    v){head[edges]=v;prv[edges]=last[u];last[u]=edges++;}
void bfs(){fill(dist,dist+n1,-1);int sizeQ=0;
  for(int u=0;u<n1;++u)if(!used[u])Q[sizeQ++]=u,dist[u]=0;
  for(int i=0;i<sizeQ;i++){int u1=Q[i];
    for(int e=last[u1];e>=0;e=prv[e]){int u2=matching[head[e]];
      if(u2>=0&&dist[u2]<0)dist[u2]=dist[u1]+1,Q[sizeQ++]=u2;}}}
bool dfs(int u1){vis[u1]=true;
  for(int e=last[u1];e>=0;e=prv[e]){
    int v=head[e],u2=matching[v];
    if(u2<0||(!vis[u2] && dist[u2]==dist[u1]+1 && dfs(u2))){
      matching[v]=u1;used[u1]=true;return true;
    }}return false;}
int maxMatching(){
  fill(used,used+n1,false);fill(matching,matching+n2,-1);
  for(int res=0;;){bfs();fill(vis,vis+n1,false);int f=0;
    for(int u=0;u<n1;++u)if(!used[u]&&dfs(u))++f;
    if(!f)return res;res+=f;}}
```

4.13 Hungarian

```
// Min cost bipartite matching.solves 1000x1000 problems.
//cost[i][j] = cost for pairing left node i with right node j
//Lmate[i] = index of right node that left node i pairs with
//Rmate[j] = index of left node that right node j pairs with
//The values in cost[i][j] may be positive or negative.
typedef vector<double>VD;typedef vector<VD>VVD;
double MinCostMatching(const VVD &cost,VI &Lmate,VI &Rmate){
  int n=SZ(cost);VD u(n);VD v(n);REP(i,n){u[i]=cost[i][0];
    for(int j=1;j<n;j++)u[i]=min(u[i],cost[i][j]);
  }REP(j,n){v[j]=cost[0][j]-u[0];
    for(int i=1;i<n;i++)v[j]=min(v[j],cost[i][j]-u[i]);
  }Lmate= VI(n, -1);Rmate = VI(n,-1);int mated=0;
  REP(i,n){REP(j,n){if(Rmate[j]!=-1)continue;
      if(fabs(cost[i][j]-u[i]-v[j])<1e-10){
        Lmate[i]=j;Rmate[j]=i;mated++;break;
      }}}VD dist(n);VI dad(n),seen(n);
  while(mated<n){int s=0;while(Lmate[s]!=-1)s++;
    fill(ALL(dad),-1);fill(ALL(seen),0);
    REP(k,n)dist[k]=cost[s][k]-u[s]-v[k];int j=0;
    while(true){j=-1;//find closest
      REP(k,n){if(seen[k])continue;if(j==-1||dist[k]<dist[j])j=k;
      }seen[j]=1;if(Rmate[j]==-1)break;
      const int i=Rmate[j];for(int k=0;k<n;k++){
        if(seen[k])continue;
        const double new_dist=dist[j]+cost[i][k]-u[i]-v[k];
        if(dist[k]>new_dist)dist[k]=new_dist,dad[k]=j;
      }}REP(k,n){if(k==j||!seen[k])continue;
      const int i=Rmate[k];v[k]+=dist[k]-dist[j];
      u[i]-=dist[k]-dist[j];}u[s]+=dist[j];
    while(dad[j]>=0){const int d=dad[j];Rmate[j]=Rmate[d];
      Lmate[Rmate[j]]=j;j=d;}Rmate[j]=s;Lmate[s]=j;mated++;
  }double val=0;REP(i,n)val+=cost[i][Lmate[i]];return val;}
```

4.14 MinCostMaxFlow

```
// Min-cost max-flow (uses SPFA. Replace by Dijkstra if needed)
// init(n)->AddEdge(x,y,cap,cost)->run(src,sink)->{flow,cost}
namespace MCMF{//MAXE = 2*calls to AddEdge
  const int MAXV = int(1e5)+10,MAXE = int(2e5)+10;
  const LL INF = 1e18;int V,E,last[MAXV],how[MAXV],adj[MAXE];
  int next[MAXE],from[MAXE];
  LL cap[MAXE],cost[MAXE],pi[MAXV],dist[MAXV];
  void init(int n){V=n;E=0;REP(i,V)last[i]=-1,pi[i]=0;}
  void AddEdge(int x, int y, LL c, LL w){
    from[E]=x;adj[E]=y;cap[E]=c;cost[E]=+w;
    next[E]=last[x];last[x]=E++;
    from[E]=y;adj[E]=x;cap[E]=0;cost[E]=-w;
    next[E]=last[y];last[y]=E++;
  }int cnt_q[MAXV],Q[MAXV],qlen;bool in_q[MAXV];
  bool SPFA(int s,int t){//replace by Dijkstra if all costs +ve
    REP(i,V)dist[i]=INF,cnt_q[i]=in_q[i]=0;
    qlen=0;Q[qlen++]=s;dist[s]=0;cnt_q[s]=1;in_q[s]=1;
    while(qlen){
      int u = Q[--qlen];in_q[u]=0;
      for(int e=last[u];e>=0;e=next[e]){
        if(cap[e]==0)continue;
        //compare dist by val in dijkstra also. rest is same
        int w=adj[e];
        LL val = dist[u]+pi[u]+cost[e]-pi[w];
        if(val>=dist[w])continue;
        dist[w]=val;how[w]=e;
        if(in_q[w])continue;
        in_q[w]=1;cnt_q[w]++;Q[qlen++]=w;
        //if(cnt_q[w]>=V)return false;
      }}return dist[t] < INF/2;}
  pair<LL, LL> run(int src, int sink){
    LL total = 0,flow = 0;
```

```
    while(SPFA(src,sink)){
      LL aug = cap[how[sink]];
      for(int i=sink;i!=src;i=from[how[i]])
        aug = min(aug,cap[how[i]]);
      for(int i=sink;i!=src;i=from[how[i]]){
        cap[how[i]]-=aug;
        cap[how[i]^1] += aug;total += cost[how[i]] * aug;
      }flow += aug;
      REP(i,V)pi[i]=min(pi[i]+dist[i],INF);
    }return make_pair(flow, total);}}
```

4.15 SCCand2SAT

```
VI order,cmpNodes[N];int vis[N],comp[N],curr;
//g:graph,rg:reverse graph
void dfs1(int u){
  vis[u]=1;
  for(auto w:g[u])
    if(!vis[w])dfs1(w);
  order.PB(u);}
void dfs2(int u){
  vis[u]=1;comp[u]=curr;cmpNodes[curr].PB(u);
  for(auto w:rg[u])if(!vis[w])dfs2(w);}
void SCC(int n){
  SET(vis,0);order.clear();
  REP(i,n)if(!vis[i])dfs1(i);
  SET(vis,0);reverse(ALL(order));curr=0;
  //components are generated in topological order
  for(auto u:order)if(!vis[u])cmpNodes[++curr].clear(),dfs2(u);
}//2-SAT : N = 2*maxvars+10,M = N/2,0-based
int val[N];int var(int x){return x<<1;}
int NOT(int x){return x^1;}
bool solvable(int vars){
  SCC(2*vars);
  REP(i,vars)if(comp[var(i)]==comp[NOT(var(i))])return false;
  return true;
}void assign_vars(){
  SET(val,0);
  for(int i=1;i<=curr;i++)
    for(auto it : cmpNodes[i]){
      int u = it>>1;
      if(val[u])continue;
      val[u] = (it&1?+1:-1);}}
void add_edge(int v1,int v2){g[v1].PB(v2);rg[v2].PB(v1);}
void add_imp(int v1,int
    v2){add_edge(v1,v2);add_edge(1^v2,1^v1);}
void add_equiv(int v1,int v2){add_imp(v1,v2);add_imp(v2,v1);}
void add_or(int v1,int
    v2){add_edge(1^v1,v2);add_edge(1^v2,v1);}
void add_xor(int v1,int v2){add_or(v1, v2);add_or(1^v1,1^v2);}
void add_true(int v1){add_edge(1^v1, v1);}
void add_and(int v1,int v2){add_true(v1);add_true(v2);}
```

4.16 SPFA

```
//Shortest Path Faster Algorithm. Computes SSSP.Works
//on graph with -ve edges.Returns false if -ve cycle.
//For -ve cycle, be careful about disconnected graphs
VII g[N];int n,cnt_q[N];bool in_q[N];
bool SPFA(int s,LL d[]){
  SET(cnt_q,0);SET(in_q,0);
  for(int i=1;i<=n;i++)d[i]=INF;
  d[s]=0;queue<int> Q;Q.push(s);
  cnt_q[s]=1;in_q[s]=1;
  while(!Q.empty()){
    int u = Q.front();Q.pop();in_q[u]=0;
    for(auto it : g[u]){
      int w=it.F,wt=it.S;
      if(d[u]+wt>=d[w])continue;
      d[w]=d[u]+wt;if(in_q[w])continue;
      in_q[w]=1;cnt_q[w]++;Q.push(w);
      if(cnt_q[w]>=n)return false;
    }}return true;}
```

4.17 StoerWagner

```
// OUTPUT:(min cut value, nodes in half of min cut)
pair<int,VI> GetMinCut(VVI &weights){//O(|V|^3)
   int N=SZ(weights),best_weight=-1;VI used(N),cut,best_cut;
  for(int phase=N-1;phase>=0;phase--){
    VI w=weights[0];VI added=used;int prev,last=0;
    REP(i,phase){prev=last;last=-1;for(int j=1;j<N;j++)
        if(!added[j]&&(last==-1||w[j]>w[last]))last=j;
      if(i==phase-1){REP(j,N)weights[prev][j]+=weights[last][j];
        REP(j,N)weights[j][prev]=weights[prev][j];
        used[last]=true;cut.push_back(last);
        if(best_weight==-1||w[last]<best_weight)
          best_cut=cut,best_weight=w[last];
      }else{REP(j,N)w[j]+=weights[last][j];added[last]=true;
      }}}return MP(best_weight,best_cut);}
```

5 MathAndDP

5.1 CHT

```
vector<LL> M,B;int ptr;// convex hull, minimum
bool bad(int a,int b,int c){//make sure LL is enough
  return (B[c]-B[a])*(M[a]-M[b])<(B[b]-B[a])*(M[a]-M[c]);
}// insert with non-increasing m
void insert(LL m, LL b){M.PB(m);B.PB(b);
  while(SZ(M) >= 3 && bad(SZ(M)-3, SZ(M)-2, SZ(M)-1)){
    M.erase(M.end()-2);B.erase(B.end()-2);
  }}LL get(int i, LL x){return M[i]*x + B[i];}
LL query(LL x){ptr=min(SZ(M)-1,ptr);
  while(ptr<SZ(M)-1 && get(ptr+1,x)<get(ptr,x))ptr++;
  return get(ptr,x);}//query with non-decreasing x
```

5.2 DivideAndConquerDP

```
LL A[N],DP[K][N],cost[N][N];int k;
void solve(int l,int r,int L,int R){
  if(l>r)return;//assuming Best[i] is monotonic
  int mid = (l+r)/2,best = L;DP[k][mid]=INF;
  for(int i = min(R,mid-1);i>=L;i--)
    if(DP[k-1][i] + cost[i+1][mid] <= DP[k][mid])
      DP[k][mid] = DP[k-1][i] + cost[i+1][mid],best = i;
  solve(l,mid-1,L,best);solve(mid+1,r,best,R);
}/*in main*/for(int i=1;i<=n;i++)DP[1][i]=cost[1][i];
for(k=2;k<=kk;k++)solve(1,n,1,n);
```

5.3 DynamicCHT

```
const LL is_query=-(1LL<<62);
struct Line{
  LL m,b;//compare two lines by increasing slope
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs)const{
    if(rhs.b!=is_query)return m<rhs.m;//> for min
    const Line* s=succ();
    if(!s)return 0;
    return b-s->b<(s->m-m)*rhs.m;}};//> for min
struct HullDynamic:public multiset<Line>{
  bool bad(iterator y){//maintains upper hull for max
    auto z=next(y);
    if(y==begin()){
      if(z==end())return 0;
      return y->m == z->m && y->b <= z->b;//>= for min
    }auto x=prev(y);
    if(z==end())
      return y->m==x->m && y->b<=x->b; // >= for min
    return (x->b-y->b)*(z->m-y->m)>=(y->b-z->b)*(y->m-x->m);
  }//Note: M * B should NOT Overflow!
  void insert_line(LL m,LL b){
    auto y=insert({ m,b});
    y->succ=[=]{return next(y)==end()?0:&*next(y);};
    if(bad(y)){erase(y);return;}
    while(next(y)!=end() && bad(next(y)))erase(next(y));
    while(y!=begin() && bad(prev(y)))erase(prev(y));
  }LL eval(LL x){
    auto l=*lower_bound((Line){x,is_query});
    return l.m*x +l.b;}};
```

5.4 FFT

```
namespace FFT{
#define op operator
  typedef long double ld;
  struct base{
    typedef double T; T re, im;
    base() :re(0), im(0) {}
    base(T re) :re(re), im(0) {}
    base(T re, T im) :re(re), im(im) {}
    base op + (const base& o)const{return base(re + o.re, im +
        o.im); }
    base op - (const base& o)const{return base(re - o.re, im -
        o.im); }
    base op * (const base& o)const{return base(re * o.re - im
        * o.im, re * o.im + im * o.re); }
    base op * (ld k) const { return base(re * k, im * k) ;}
    base conj() const { return base(re, -im); }
  };const int N = 21;const int MAXN = (1<<N);
  const double PI = acos(-1);
  base w[MAXN];base f1[MAXN];int rev[MAXN];
  void build_rev(int k){
    static int rk = -1;
    if( k == rk )return ; rk = k;
    FOR(i,1,(1<<k)+1){
      int j = rev[i-1], t = k-1;
      while(t >= 0 && ((j>>t)&1) ) { j ^= 1 << t; --t; }
      if(t >= 0) { j ^= 1 << t; --t; }
      rev[i] = j;}}
  void fft(base *a, int k) {
```

```
  build_rev(k); int n = 1 << k;
  REP(i, n) if( rev[i] > i ) swap(a[i], a[rev[i]]);
  for(int l = 2, ll = 1; l <= n; l += l, ll += ll) {
    if( w[ll].re == 0 && w[ll].im == 0 ) {
      ld angle = PI / ll;
      base ww( cosl(angle), sinl(angle) );
      if( ll > 1 ) for(int j = 0; j < ll; ++j) {
        if( j & 1 ) w[ll + j] = w[(ll+j)/2] * ww;
        else w[ll + j] = w[(ll+j)/2];
      } else w[ll] = base(1, 0);
    }
    for(int i = 0; i < n; i += l) REP(j, ll) {
      base v = a[i + j], u = a[i + j + ll] * w[ll + j];
      a[i + j] = v + u; a[i + j + ll] = v - u; }}}
void mult(LL *a,LL *b,LL *c,int len){
  int k = 1;while((1<<k) < (2*len)) ++k;int n = (1<<k);
  REP(i, n)f1[i] = base(0,0);
  REP(i, len)f1[i] = f1[i] + base(a[i], 0);
  REP(i, len)f1[i] = f1[i] + base(0, b[i]);
  fft(f1, k);
  REP(i, 1 + n/2) {
    base p = f1[i] + f1[(n-i)%n].conj();
    base _q = f1[(n-i)%n] - f1[i].conj();
    base q(_q.im, _q.re);
    f1[i] = (p * q) * 0.25;
    if( i > 0 ) f1[(n - i)] = f1[i].conj();
  }REP(i, n) f1[i] = f1[i].conj();
  fft(f1, k);
  REP(i, 2*len){
    c[i] = LL(f1[i].re / n + 0.5);
  }}/*slow mult. faster to code. ignore above part*/
typedef complex<double> base;
base omega[MAXN],a1[MAXN],a2[MAXN],z1[MAXN],z2[MAXN];
void fft(base *a, base *z, int m = N){
  if (m==1)z[0] = a[0];
  else{int s=N/m; m /= 2;
    fft(a,z,m);fft(a+s,z+m,m);
    REP(i, m){base c = omega[s*i] * z[m+i];
      z[m+i] = z[i] - c;z[i] += c;}}}
void mult(LL *a,LL *b,LL *c,int len){
  N = 2*len;while (N & (N-1)) ++N;assert(N <= MAX);
  REP(i, N) a1[i] = 0;REP(i, N) a2[i] = 0;
  REP(i, len) a1[i] = a[i];REP(i, len) a2[i] = b[i];
    REP(i, N) omega[i] = polar(1.0, 2*PI/N*i);
    fft(a1, z1, N);fft(a2, z2, N);
    REP(i, N) omega[i] = base(1, 0) / omega[i];
    REP(i, N) a1[i] = z1[i] * z2[i] / base(N, 0);
    fft(a1, z1, N);
    REP(i, 2*len) c[i] = round(z1[i].real());}}
    void mul_mod(LL *a, LL *b, LL *c, int len, const int mod){
      static LL a0[MAXN],a1[MAXN],b0[MAXN],b1[MAXN];
      static LL c0[MAXN],c1[MAXN],c2[MAXN];
      REP(i, len) a0[i] = a[i] & 0xFFFF;
      REP(i, len) a1[i] = a[i] >> 16;
      REP(i, len) b0[i] = b[i] & 0xFFFF;
      REP(i, len) b1[i] = b[i] >> 16;
      mult(a0, b0, c0, len);mult(a1, b1, c2, len);
      REP(i, len) a0[i] += a1[i];
      REP(i, len) b0[i] += b1[i];
      mult(a0, b0, c1, len);
      REP(i, 2*len)c1[i] -= c0[i] + c2[i];
      REP(i, 2*len)c1[i] %= mod;
      REP(i, 2*len)c2[i] %= mod;
      REP(i, 2*len)c[i]=(c0[i]+(c1[i]<<16)+(c2[i]<<32))%mod;
    }}// end of FFT namespace
//For solving recurrences of the form F_i=sum(1 <=j<i)F_j*G_n-j
void convolve(int l1, int r1, int l2, int r2){
  A = F[l1 .. r1]; B = G[l2 ... r2];//0-based polynomials
  C = A * B;//multiplication of two polynomials.
  for(int i = 0; i < C.size(); ++i)
    F[l1 + l2 + i] += C[i];
}//in main function.
F[1] = 1;//some base case.
for(int i = 1;i <= n - 1; i++){
  //We have computed till F_i and want to add its contribution.
  F[i + 1] += F[i] * G[1]; F[i + 2] += F[i] * G[2];
  for(int pw = 2; i % pw == 0 && pw + 1 <= n; pw = pw * 2){
    //iterate over every power of 2 untill 2 ^ i divides i.
    convolve(i - pw, i - 1, pw + 1, min(2 * pw, n));}}
```

5.5 Fibonacci

```
//using f(a+b)=f(a+1)f(b)+f(b-1)f(a);
LL fib(LL n,LL mod){
  LL i,h,j,k,t;i=h=1;j=k=0;
  while(n>0){if(n%2==1){t=(j*h)%mod;
    j=(i*h + j*k +t)%mod;i=(i*k + t)%mod;}
```

```
    t=(h*h)%mod;h=(2*k*h + t)%mod;
    k=(k*k + t)%mod;n=n/2;}return j;
}LL pisano(int mod){ LL period=1,i;
  for(i=2;i*i<=mod;i++){if(mod%i==0){
    if(i==2) period*=3;else if(i==5)
    period*=20;else period*=(i-1)*(i+1);
    mod/=i;while(mod%i==0){period*=i,mod/=i;}
  }if(mod>1){i=mod;if(i==2)period*=3;
  else if(i==5)period*=20;
  else period*=(i-1)*(i+1);}return period;}}
```

5.6 GaussModP

```
// Solves systems of linear modular equations.mat[i][C]=b[i];
// Build a matrix of coefficients and call run(mat, R, C, mod).
// If no solution,returns -1, else returns # of free variables.
// If i-th variable free,row[i]=-1,else it's value = ans[i].
// Time complexity: O(R * C^2) - MAXC is the number of columns
namespace Gauss{const int MAXC=1001;int row[MAXC];LL ans[MAXC];
  LL inv(LL x,LL mod){return power(x,mod-2,mod);}
  int run(LL mat[][MAXC],int R,int C,LL mod){REP(i,C)row[i]=-1;
    int r=0;REP(c,C){int k=r;while(k<R && mat[k][c]==0)++k;
      if(k==R)continue;REP(j,C+1)swap(mat[r][j],mat[k][j]);
      LL div=inv(mat[r][c],mod);REP(i,R)if(i!=r){
        LL w = mat[i][c]*(mod-div)%mod;
        REP(j,C+1) mat[i][j]=(mat[i][j]+mat[r][j]*w)%mod;
      }row[c] = r++;}REP(i,C){int r = row[i];
        ans[i]=(r==-1?0:mat[r][C])*inv(mat[r][i],mod)%mod;
      }FOR(i, r, R)if(mat[i][C])return -1;return C - r;}}
namespace GaussMod2{//Every x in basis has leftmost bit 1 s.t
  void add(int x){//every y!=x has that bit=0.Rank=SZ(basis)
    for(auto &y : basis)if((y ^ x) < x)x ^= y;
    for(auto &y : basis)if((y ^ x) < y)y ^= x;
    if(x)basis.PB(x), sort(ALL(basis));
  }int query(int k){k--;//kth smallest xor.1 based.
    int ret=0;REP(i,SZ(basis))if((1<<i)&k)ret ^= basis[i];
    return ret;}}
```

5.7 MatrixOperations

```
// Gauss-Jordan elimination solves (AX = B). O(n^3)
// INPUT:   a[][] = an nxn matrix
//          b[][] = an nxm matrix
// OUTPUT:  X     = an nxm matrix (stored in b[][])
//          A^{-1} = an nxn matrix (stored in a[][])
//          returns determinant of a[][]
const double EPS = 1e-10;
typedef double T;typedef vector<T> VT;typedef vector<VT> VVT;
T GaussJordan(VVT &a, VVT &b) {
  const int n=a.size(),m=b[0].size();
  VI irow(n),icol(n),ipiv(n);T det=1;
  for(int i=0;i<n;i++){
    int pj=-1,pk=-1;
    for(int j=0;j<n;j++)if(!ipiv[j])
      for(int k=0;k<n;k++)if(!ipiv[k])
        if(pj==-1||fabs(a[j][k])>fabs(a[pj][pk])){pj=j;pk=k;}
    if(fabs(a[pj][pk])<EPS){cerr<<"Matrix is
        singular."<<endl;exit(0);}
    ipiv[pk]++;swap(a[pj], a[pk]);swap(b[pj],b[pk]);
    if(pj!=pk)det*=-1;irow[i]=pj;icol[i]=pk;
    T c =1.0/a[pk][pk];det*=a[pk][pk];a[pk][pk]=1.0;
    for(int p=0;p<n;p++)a[pk][p]*=c;for(int
        p=0;p<m;p++)b[pk][p]*=c;
    for(int p=0;p<n;p++)if(p!=pk){c=a[p][pk];a[p][pk]=0;
      for(int q=0;q<n;q++)a[p][q]-=a[pk][q]*c;
      for(int q=0;q<m;q++)b[p][q]-=b[pk][q]*c;
    }}for(int p=n-1;p>=0;p--)if(irow[p]!=icol[p])
    for(int k=0;k<n;k++)swap(a[k][irow[p]],a[k][icol[p]]);
  return det;}
```

5.8 MatrixRank

```
// INPUT:   a[][] = an nxm matrix. (rref : reduced row echelon)
// OUTPUT:  rref[][] = an nxm matAix(stored in a[][]),
const double EPS =1e-10;// returns rank of a[][]
typedef double T;typedef vector<T> VT;typedef vector<VT> VVT;
int rref(VVT &a){
  int n=a.size(),m=a[0].size(),r=0;
  for (int c=0;c<m&&r<n;c++){int j=r;
    for(int i=r+1;i<n;i++)if(fabs(a[i][c])>fabs(a[j][c]))j=i;
    if(fabs(a[j][c])<EPS)continue;swap(a[j],a[r]);
    T s=1.0/a[r][c];for(int j=0;j<m;j++)a[r][j]*=s;
    for(int i=0;i<n;i++)if(i!=r){T t=a[i][c];
      for(int j=0;j<m;j++)a[i][j]-=t*a[r][j];
    }r++;}return r;}
```

5.9 NumberTheory

```
tuple<LL,LL,LL> extended_euclid(LL a,LL b){
  LL s=0,ss=1,t=1,tt=0,r=b,rr=a,tmp;while(r){tmp=ss-(rr/r)*s;
    ss=s;s=tmp;tmp=tt-(rr/r)*t;tt=t;t=tmp;tmp=rr%r;rr=r;r=tmp;
  }if(a<0){ss=-ss;tt=-tt;rr=-rr;}//ss*a+tt*b=rr=gcd(a,g)
```

```
  return make_tuple(ss,tt,rr);
}LL mod(LL a,LL N){a%=N;return a<0?a+N:a;}
LL modmul(LL a,LL b,LL
    N){a=mod(a,N);b=mod(b,N);if(a<b)swap(a,b);
LL res=0;for(int i=63-__builtin_clzll(b);i>=0;--i){
res=(res+res)%N;if((b>>i)&1)res=(res+a)%N;}return res;}
LL modpow(LL b,LL e,LL N){LL res=1;
for(int
    i=63-__builtin_clzll(e);i>=0;--i){res=modmul(res,res,N);
    if((e>>i)&1)res=modmul(res,b,N);}return res;
}LL mod_inverse(LL a,LL n){LL b,k,g;//ba+kn=gcd(a, n)
  tie(b,k,g)=extended_euclid(a,n);return (g!=1?-1:mod(b,n));
}//crt for n tems can be found by iterating over n terms.
pair<LL,LL> chinese_remainder_theorem(LL x,LL a,LL y,LL b){
  //finds z (mod M) so z=a (mod x) and z=b (mod y),lcm
  LL s,t,d;tie(s,t,d)=extended_euclid(x,y);
  if(a%d!=b%d)return make_pair(0,-1);LL M=x*y;
  LL z=(modmul(modmul(s,b,M),x,M)+modmul(modmul(t,a,M),y,M))%M;
  return make_pair(z/d,M/d);}//returns x,y such that c=ax+by
pair<LL,LL> linear_diophantine(LL a,LL b,LL c){
  LL d=__gcd(a,b);if(c%d!=0)return make_pair(-1,-1);
  return make_pair((c/d)*mod_inverse(a/d,b/d),(c-a*x)/b);
}//returns all solutions to ax=b mod n
vector<int> modular_linear_equation_solver(int a,int b,int n){
  LL x,y,d;tie(x,y,d)=extended_euclid(a,n);vector<int> ans;
  if(b%d==0){b/=d;n/=d;x=mod(x*b,n);for(LL i=0;i<d;++i)
    ans.push_back(mod(x+i*n,n));}return ans;
}bool miller_rabin_primality(LL N){
  //deterministic for all<=2 ^ 64
  static const int p[12]={2,3,5,7,11,13,17,19,23,29,31,37};
  if(N<=1)return false;for(int i=0;i<12;++i){
    if(p[i]==N)return true;if(N%p[i]==0)return false;
  }LL c=N-1,g=0;while(!(c&1))c>>=1,++g;
  for(int i=0;i<12;++i){LL k=modpow(p[i],c,N);
    for(int j=0;j<g;++j){LL kk=modmul(k,k,N);
      if(kk==1&&k!=1&&k!=N-1)return false;k=kk;}
    if(k!=1)return false;}return true;
}mt19937 gen(time(0));//gives a factor of N
LL pollard_rho(LL N){if(N%2==0)return 2;
  LL xx=uniform_int_distribution<LL>()(gen)%N,x=xx;
  LL c=uniform_int_distribution<LL>()(gen)%N,d=1;
  for(int iters=0;iters<2000;++iters){
    x=(modmul(x,x,N)+c)%N;xx=(modmul(xx,xx,N)+c)%N;
    xx=(modmul(xx,xx,N)+c)%N;d=__gcd(abs(x-xx),N);
    if(d!=1&&d!=N)break;}return d;}
#define M(x) x%p//solves a^2=x(mod p),return -1 if x not exist
LL root_of_x(LL x,LL p){LL r=0,s=p-1,n,m,x,b,g,coff,t;
  if(power(a,((p-1)>>1),p)==p-1)return -1;
  //calcute (a^((p-1)/2))%p;
  while((s&1)==0){s=(s>>1);r++;}for(LL i=2;i<p;i++)
    if(power(i,((p-1)>>1),p)==p-1){n=i;break;}
  b=power(a,s,p);g=power(n,s,p);x=power(a,((s+1)>>1),p);
  while(r>0){t=b;for(m=0;m<r;m++){if(M(t)==1)break;t=M(t*t);}
    if(m>0){coff=power(g,(1<<(r-(m+1))),p);x=M(x*coff);
      g=M(coff*coff);b=M(b*g);}r=m;}return x;}
//To factorize in N^(1/3),do normal+miller_rabin+pollard_rho
//to get the remaining prime x s.t. x * x or x and n / x.
```

5.10 SOSDP

```
//Given an array A of 2^N integers,calculate  x, F(x) = Sum of
//all A[i] such that x & i = i, i.e., i is a subset of x.
//dp[mask][i]:subsets where first i bits of mask can differ
REP(mask,(1<<N)){dp[mask][-1] = A[mask];//handle base case
  REP(i,N)if(mask&(1<<i))
      dp[mask][i] = /*-1*/dp[mask][i-1]+dp[mask^(1<<i)][i-1];
    else dp[mask][i] = dp[mask][i-1];//for MU (invers) -1 will
  }F[mask] = dp[mask][N-1];//come there.
}//memory optimized, super easy to code.
REP(i,(1<<N))F[i] = A[i];REP(i,N)REP(mask,(1<<N))
  if(mask&(1<<i))F[mask]+=F[mask^(1<<i)];
//note : to iterate over submasks of a mask use this
for(int s=m; s; s=(s-1)&m) //process 0 separately
//Zeta = SOS, MU = (-1) ^ |S / S'|, MU . Z = I,
//Sigma F(S) = (-1)^|S| F(S). MU F = Sigma . Z . Sigma F,
//F'(k, X) : Sum of all subsets of size k of X. Compute in
//O(n^2 2 ^ n) for every k by setting remaining 0 coz ind.
//F * G : Subset Convolution = MU(F o G(K, X)).
//F o G(K, X) = SUM(j <= k) f'(j, X) x g'(k - j, X) : k^2 M
```

5.11 Simplex

```
//maximize c^T x (T-->transpose)subject to Ax<=b,x >= 0
//INPUT,A:mxn matrix,b:1*m vect,c:1*n vect,x:ans vect,
//OUTPUT,opt soln(infinity:unbounded above/nan:infeasible)
//.LPS Object:A,b,and c as args.Then,call Solve(x).
//typedef long double ld,vector<ld> VD, vector<VD> VVD;
const ld EPS=1e-9,inf=numeric_limits<ld>::infinity();
struct LPSolver{int m,n;VI B,N;VVD D;
```

```
LPSolver(const VVD &A,const VD &b,const VD &c):
  m(b.size()),n(c.size()),N(n+1),B(m),D(m+2,VD(n+2)){
    REP(i,m)REP(j,n)D[i][j]=A[i][j];REP(i,m){B[i]=n+i;
    D[i][n]=-1;D[i][n+1]=b[i];}REP(j,n){N[j]=j;
    D[m][j]=-c[j];}N[n]=-1;D[m+1][n]=1;}
void Pivot(int r,int s){REP(i,m+2)if(i!=r)REP(j,n+2)if(j!=s)
  D[i][j]-=D[r][j]*D[i][s]/D[r][s];REP(j,n+2)if(j!=s)
  D[r][j]/=D[r][s];REP(i,m+2)if(i!=r)D[i][s]/=-D[r][s];
  D[r][s]=1.0/D[r][s];swap(B[r],N[s]);}
bool Simplex(int phase){int x=(phase==1?m+1:m);while(true){
  int s=-1;REP(j,n+1){if(phase==2&&N[j]==-1)continue;
    if(s==-1||D[x][j]<D[x][s]||D[x][j]==D[x][s]&&N[j]<N[s])s=j;
  }if(D[x][s]>=-EPS)return true;int r=-1;REP(i,m){
    if(D[i][s]<=0)continue;
    if(r==-1||D[i][n+1]/D[i][s]<D[r][n+1]/D[r][s]||
        D[i][n+1]/D[i][s]==D[r][n+1]/D[r][s]&&B[i]<B[r])r=i;
  }if(r==-1)return false;Pivot(r,s);}}
ld Solve(VD &x){int r=0;FOR(i,1,m)if(D[i][n+1]<D[r][n+1])r=i;
  if(D[r][n+1]<=-EPS){Pivot(r,n);if(!Simplex(1)||D[m+1][n+1]<-EPS)
    return -inf;REP(i,m)if(B[i]==-1){int s=-1;REP(j,n+1)
    if(s==-1||D[i][j]<D[i][s]||D[i][j]==D[i][s]&&N[j]<N[s])s=j;
    Pivot(i,s);}}if(!Simplex(2))return inf;x=VD(n);REP(i,m)
      if(B[i]<n)x[B[i]]=D[i][n+1];return D[m][n+1];};
```

5.12 SimpsonsMethod

```
#define T double // Simpson rule. integration of f from a to b
T f(T x){return x*x;}const T eps=1e-12;
T simps(T a,T b,T fa,T fm,T fb){return (fa+4*fm+fb)*(b-a)/6;}
T integr(T a,T b,T fa,T fm,T fb){T m=(a+b)/2;T fam=f((a+m)/2),
  fmb=f((m+b)/2);T l=simps(a,m,fa,fam,fm),r=simps(m,b,fm,fmb,
  fb),tot=simps(a,b,fa,fm,fb);if(fabs(l+r-tot)<eps)return tot;
  return integr(a,m,fa,fam,fm) + integr(m,b,fm,fmb,fb);}
T integrate(T a, T b){return integr(a,b,f(a),f((a+b)/2),f(b));}
```

5.13 XorConvolution

```
void convolve(LL P[], int N, bool inverse){
  for(int i = 2 ; i <= N ; i <<= 1){
    int m = i >> 1;
    int u , v , x , y;
    for(int j = 0 ; j < N ; j += i){
      for(int k = 0 ; k < m ; ++k){
        u = P[j + k];
        v = P[j + k + m];
        if(!inverse){
          P[j + k] = c0 * u + c1 * v;
          P[j + k + m] = c2 * u + c3 * v;
        }else {
          P[j + k] = d0 * u + d1 * v;
          P[j + k + m] = d2 * u + d3 * v;
        } }}}
}//For XOR, divide inverse by n finally.
//XOR: C = [+1, +1, +1, -1], D = [+1, +1, +1, -1]
//AND: C = [+0, +1, +1, +1], D = [-1, +1, +1, +0]
//OR : C = [+1, +1, +1, +0], D = [+0, +1, +1, -1]
```

5.14 nCrLarge

```
LL invert_mod(LL k,LL m){
  if(m==0)return(k==1||k==-1)?k:0;if(m<0)m=-m;k%=m;
  if(k<0)k+=m;int neg=1;LL p1=1,p2=0,k1=k,m1=m,q,r,temp;
  while(k1>0){q=m1/k1;r=m1%k1;temp=q*p1+p2;p2=p1;p1=temp;
    m1=k1;k1=r;neg=!neg;}return neg?m-p2:p2;
}// Preconditions:0<=k<=n;p>1 prime
LL choose_mod_one(LL n,LL k,LL p){
  if(k<p)return choose_mod_two(n,k,p);
  LL q_n,r_n,q_k,r_k,choose;
  q_n=n/p;r_n=n%p;q_k=k/p;r_k=k%p;
  choose=choose_mod_two(r_n,r_k,p);
  choose*=choose_mod_one(q_n,q_k,p);
  return choose%p;
}// Preconditions:0<=k<=min(n,p-1);p>1 prime
LL choose_mod_two(LL n,LL k,LL p){
  n%=p;if(n<k)return 0;if(k==0||k==n)return 1;
  if(k>n/2)k=n-k;LL num=n,den=1;
  for(n=n-1;k>1;--n,--k)num=(num*n)%p,den=(den*k)%p;
  den=invert_mod(den,p);return (num*den)%p;
}LL fact_exp(LL n, LL p){LL ex=0;do{n/=p;ex+=n;
  }while(n>0);return ex;
}//returns nCk % p in O(p).n and k can be large.
LL choose_mod(LL n, LL k, LL p){
  if(k<0||n<k)return 0;if(k==0||k==n)return 1;
  if(fact_exp(n)>fact_exp(k)+fact_exp(n-k))return 0;
  return choose_mod_one(n,k,p);}
```

6 String

6.1 AhoCorasick

```
namespace AhoCorasick{
  const int MAXN = int(1e5)+10;//pnode out[MAXN];
```

```
  map<char,int> to[MAXN];int f[MAXN],blen;bool end[MAXN];
  void add_str(int idx,string &s){int x = 0;
    for(auto c : s){if(!to[x][c])to[x][c] = ++blen;
      x = to[x][c];}/*insert(out[x],idx);*/end[x] = true;
  }int next(int state,char c){
    while(state && !to[state].count(c))state = f[state];
    return to[state][c];
  }void build_SL(){queue<int> Q;
    for(auto it : to[0])if(it.S)Q.push(it.S);
    while(!Q.empty()){int x = Q.front();Q.pop();
      for(auto it : to[x]){int y = it.S, c = it.F;
        f[y] = next(f[x],c);Q.push(y);}
      /*merge(out[x],out[f[x]]);*/end[x] |= end[f[x]];
    }}VII findAllOccurences(string &s){int x = 0;VII ret;
    for(int i=0;i<SZ(s);i++){char c = s[i]; x = next(x,c);
      for(pnode it = out[x]; it != NULL; it = it->nxt)
        ret.PB({i,it->val});}return ret;}}
```

6.2 KMP

```
int nfa[N];//preprocess pattern & search in any text in O(|T|)
  void build_NFA(string &P){
    nfa[0]=0;int x=0,n=SZ(P);
    for(int i=1;i<=n;i++){
      nfa[i]=x;
      while(i!=n){
        if(P[x]==P[i]){x++;break;}
        if(!x)break;x=nfa[x];}}}
int kmp_search(int start,string& P,string& T){
  for(int i=start,x=0;i<SZ(T);){
    if(T[i]==P[x])x++,i++;
    else if(!x)i++;
    else x = nfa[x];
    if(x==SZ(P))return i-SZ(P);
  }return -1;
}//ans=kmp_search(ans),ans+=(SZ(P)-nfa[SZ(p)])
```

6.3 PalindromicTree

```
const int N = int(1e5)+10;
struct node{
  int nxt[26];//edge u-x->v: v = xux where x : character.
  int len;//length of palindrome stored at this node.
  int sufflink;//u-s->v,v:longest proper palindromic suff of u.
  int s,e;//start,end indices of palindrome in string
  int num;//No of reachable suffix links by the current node.
  node(){SET(nxt,0);len=sufflink=s=e=0;}
}tree[N];//all nodes of tree. Buffer.blen : buffer length
int blen;//node 1:root with len -1,2:root with len 0
int suff;//Maximum suffix palindrome at any point of time.
bool addLetter(string & s,int pos){//true=new node is created
  int cur=suff,curlen=0,idx=s[pos]-'a';
  while(1){//Find L.S. A s.t. xAx is new LS Palindrom
    curlen = tree[cur].len;//start with cur.
    if(pos-1-curlen>=0&&s[pos-1-curlen]==s[pos])break;
    cur=tree[cur].sufflink;//LS : Longest Suffix
  }//If node for xAx already exists
  if(tree[cur].nxt[idx]){//set new suff = index of already...
    suff=tree[cur].nxt[idx];return false;//...existing node xAx
  }suff = ++blen;//add new node for xAx and update suff
  tree[blen].len=tree[cur].len+2;//length of xAx.starting & ...
  tree[blen].s=pos-1-curlen;tree[blen].e=pos;//ending indices..
  tree[cur].nxt[idx]=blen;//of xAx.on adding x to A,we get xAx.
  //Now we need to search for the suffix link of newly formed..
  if(tree[blen].len==1){//...node blen i.e. xAx
    tree[blen].sufflink=2;tree[blen].num=1;//if xAx == x,
    return true;//set suffix link equal to empty string(node 2)
  }//else search for the suffix link
  while(1){//Initially cur-->A. Find LS B of A s.t. xBx
    cur=tree[cur].sufflink;curlen=tree[cur].len;
    if(pos-1-curlen>=0 && s[pos-1-curlen]==s[pos]){
      //Found B.Could be -1 s.t. sufflink to node 1
      tree[blen].sufflink=tree[cur].nxt[idx];
      break;//Set the sufflink and we are done.
    }//O(N) coz finding new suff can move left pointer to one
  }//unit left and following suffix links always moves it to
  tree[blen].num=tree[tree[blen].sufflink].num+1;//right. So,
  return true;//it can travel at most O(2*N) times to right.
}void initTree(){//Initialize the tree.
  blen=2;suff=2;//Longest suffix initially is the empty string.
  tree[1].len=-1;tree[1].sufflink=1;//Node 1-->root with len -1
  tree[2].len=0;tree[2].sufflink=1};//Node 2-->root with len 0
```

6.4 SuffixArray

```
//LCP[0][i]= len(LCP) of SA[i] & SA[i+1](sorted suffixes).
//RA[i][j] = Rank of suffix S[j...j+2^i]
//SA[i] = i'th Lexicographically smallest suffix's index.
int RA[LOGN][N],SA[N],tempSA[N],cnt[N],msb[N];
int LCP[LOGN][N],dollar[N];
```

```
void countingSort(int l,int k,int n){
  SET(cnt,0);
  for(int i=0;i<n;i++){
    int idx=(i+k<n?RA[l][i+k]:0);cnt[idx]++;
  }int maxi=max(300,n);
  for(int i=0,sum=0;i<maxi;i++){
    int t=cnt[i];cnt[i]=sum;sum+=t;}
  for(int i=0;i<n;i++){
    int idx = SA[i]+k<n?RA[l][SA[i]+k]:0;
    tempSA[cnt[idx]++]=SA[i];
  }for(int i=0;i<n;i++)SA[i]=tempSA[i];
}void build_SA(string &s){
  int n = SZ(s);
  for(int i=0;i<n;i++)RA[0][i]=s[i];
  for(int i=0;i<n;i++)SA[i]=i;
  for(int i=0;i<LOGN-1;i++){
    int k = (1<<i);if(k>=n)break;
    countingSort(i,k,n);countingSort(i,0,n);
    int rank=0;RA[i+1][SA[0]]=rank;
    for(int j=1;j<n;j++)
      if(RA[i][SA[j]]==RA[i][SA[j-1]]&&
      RA[i][SA[j]+k]==RA[i][SA[j-1]+k])RA[i+1][SA[j]]=rank;
      else RA[i+1][SA[j]]=++rank;
  }}void build_msb(){int mx=-1;
  for(int i=0;i<N;i++){if(i>=(1<<(mx+1)))mx++;msb[i]=mx;}
}void build_LCP(string& s){
  int n =SZ(s);
  for(int i=0;i<n-1;i++){//Build the LCP array in O(NlogN)
    int x = SA[i],y=SA[i+1],k,ret=0;
    for(k=LOGN-1;k>=0 && x<n && y<n;k--){
      if((1<<k)>=n)continue;
      if(RA[k][x]==RA[k][y])x+=1<<k,y+=1<<k,ret+=1<<k;
    }if(ret>=dollar[SA[i]]-SA[i])ret=dollar[SA[i]]-SA[i];
    LCP[0][i]=ret;//LCP[i] shouldn't exceed dollar[SA[i]]
  }//dollar[i] : index of dollar to the right of i.
  LCP[0][n-1]=10*N;
  for(int i=1;i<LOGN;i++){//O(1) RMQ structure in O(NlogN)
    int add = (1<<(i-1));if(add>=n)break;//small optimization
    for(int j=0;j<n;j++)
      if(j+add<n)LCP[i][j] = min(LCP[i-1][j],LCP[i-1][j+add]);
      else LCP[i][j] = LCP[i-1][j];
  }}int lcp(int x,int y){
  //O(1) LCP.x & y are indexes of the suffix in SA!
  if(x==y)return dollar[SA[x]]-SA[x];if(x>y)swap(x,y);y--;
  int idx=msb[y-x+1],sub=(1<<idx);
  return min(LCP[idx][x],LCP[idx][y-sub+1]);
}bool equal(int i,int j,int p,int q){
  if(j-i!=q-p)return false;
  int idx=msb[j-i+1],sub=(1<<idx);
  return RA[idx][i]==RA[idx][p] &&
      RA[idx][j-sub+1]==RA[idx][q-sub+1];
}//Note : Do not forget to add a terminating '$'
```

6.5 SuffixAutomation

```
struct SA{//u & w are endpos equiv iff u is found in s only
  VVI to;VI dp,link,len,val;//as a suffix of string w.
  int last,sz;//if len(u) <= len(w), then either enpos(w) is a
  SA(){//subset of endpos(u) or they both are disjoint sets.
    to.clear();dp.clear();link.clear();len.clear();val.clear();
    last = sz = 0;//len[i] : length of the longest substring
  }//belonging to the equivalence class of state/node i.
  SA(int n){//link[i]:suffix link from state i leading to state
    to = VVI(n,VI(sigma,0)); dp = link = len = val = VI(n,0);
    last = sz = 0;//corresponding to a suffix of longest(i)
  }//of length minlen(i)-1. Therefore minlen(i)=len[link[i]]+1
  SA(string &s){//suffix links form a tree in which endpos of
    int n=2*SZ(s)+10;to=VVI(n,VI(sigma,0));//child is subset
    dp=link=len=val=VI(n,0);last=sz=0;//of endpos of its
    for(auto c:s)add_letter(c-'a');go();//parent in the tree.
  }//Substrings of length [minlen(i),len(i)] ending in index
  void add_letter(int c){//pos(i):first index in the string s
    int p = last;//at which substrings belonging to state i end
    last = ++sz;//last:state of s.Create state for string sc.
    len[last]=len[p]+1;//also set pos[last] = currindex here.
    for(;!to[p][c];p=link[p])to[p][c]=last;
    //check if it's the first occurence of c in the string
    if(to[p][c]==last)return void(link[last]=0);//if yes,
    int q = to[p][c];//return. else check if a solid transition
    if(len[q] == len[p] + 1)return void(link[last]=q);
    int x = ++sz;//we need to produce a clone of the state q.
    to[x] = to[q];//also copy pos[x]=pos[q]. x is a clone node.
    link[x]=link[q];len[x]=len[p]+1;//update len for x.
    link[last]=link[q]=x;//update suffix links of q and last.
    for(;to[p][c]==q;p=link[p])to[p][c]=x;//update transitions
  }//dp to compute no. of terminal nodes reachable from node x.
  void f(int x){//Assuming val[x]:represents whether x is a
```

```
    if(dp[x])return;//terminal node or not. Usual dag dp.
    dp[x] = val[x];//Other DAG algo's can also be applied acc
    for(int i=0;i<sigma;i++)//to the problem. Like number of
      if(to[x][i])//vertex/edge disjoint paths from source to
        f(to[x][i]),dp[x]+=dp[to[x][i]];//sink etc.
  }//mark the terminal nodes and compute the dp. To mark the
  void go(){//terminal nodes go over all the suffix links of
    for(int x=last;x;x=link[x])val[x]+=1;//the last node coz
    f(0);//last corresponds to state repesenting whole string
  }//run the query string through the automation.
  int get(string &s){//Also tree formed by suffix links is same
    int x = 0;//as the suffix tree of the reverse of string s.
    for(auto cc:s){//to report all occurences of P in S, build
      int c = cc-'a';//SA of S and run P. If it ends at state x
      if(!to[x][c])return 0;//Do a dfs from x in the rev graph
      else x = to[x][c];//of suffix links i.e. if link[x]=y,
    }return dp[x];//then g[y].PB(x).For each visited state,add
  }};//pos[i] to ans. To avoid repitition if node i is a
//cloned node,ignore it else add pos[i] to ans. O(|P|+|output|)
```

6.6 SuffixTree

```
namespace SuffixTree{//O(Nlog(Sigma)) construction & use.
  const int INF = 1e9,N = 1e6 + 10,dollar = 257;//set
  int s[N];//dollar = 1 + MaxSigma. Ascii in this case
  map<int, int> to[N];//to[from]:(char,to_node).Just like
  int len[N]={INF},fpos[N],link[N],suff[N],par[N];//tries
  int node, pos, remain;//root node = 0.par[node]:mainly
  int sz = 1, n = 0;//valid for leaf nodes.(not much use).
  int make_node(int _pos, int _len,int _par){//fpos[node]:
    fpos[sz] = _pos;len[sz] = _len;//the leftmost index of
    suff[sz] = n - remain;par[sz] = _par;//the substring
    return sz++;//represented by the parent edge of node.
  }//len[node]:stores the length of the subtring of parent
  void go_edge(){//edge of the node.Note that for leaves,
    while(pos > len[to[node][s[n - pos]]]){//len[node]=INF.
      node = to[node][s[n - pos]];//The [L,R] of the parent
      pos -= len[node];//edge of a node can be computed as:
    }}//L=fpos[y],R=min(n-1,l+len[y]-1),where n=strlen(s).
  void add_letter(int c){//suff[node]:represents what is
    s[n++] = c;int last = 0;//the index of the suffix which
    for(remain++,pos++;pos > 0;remain--){//ends at this node.
      go_edge();//useful only for leaves, and when inserting
      int edge = s[n - pos];//concatenation of many strings
      int &v = to[node][edge];//separeted by (dollar + i)
      int t = s[fpos[v] + pos - 1];//distinct separators.
      if(v == 0){//In case of multiple strings, note that
        v = make_node(n - pos, INF,node);//a dollar can
        link[last] = node;//occur on an edge,so be careful
        last = 0;//while traversing such edges. In general,
      }else if(t == c){//do not traverse an edge having a
        link[last] = node;//dollar and add it's contribution
        return;//there itself coz only one usefull suffix
      }else{//will be there in ths subtree of this node
        int u = make_node(fpos[v], pos - 1,node);//whose
        to[u][c] = make_node(n - 1, INF,u);//parent edge has
        to[u][t] = v;par[v] = u;//a dollar in it. This is so
        fpos[v] += pos - 1;//coz any suffix staring at < l
        len [v] -= pos - 1;//would have traversed at least
        v = u;//one more dollar sign before, which is not
        link[last] = u;//possible since this is the first
        last = u;//encountered dollar & any suffix starting
      }//at > r cannot traverse this dollar since all dollars
      if(node == 0) pos--;//are distinct. Only one suffix can
      else node = link[node];//start from [l,r] and be here
    }//coz different suffixes will have different lengths.
  }//In general,be careful about adding contributions of egdes.
}//Ex:for distinct bracket substrings, check if a bracket seq
//ends on this edge,and if yes, where all?go forward only if
//current prefix sum >=0.BTW use ST only if you need to :P
```

6.7 ZAlgo

```
int L=0,R=0;//compute Z array s.t. Z[i] stores length of the
for(int i=1;i<n;i++){
  if(i>R){
    L=R=i;//longest substring starting
    while(R<n&&s[R-L]==s[R])R++;//from S[i] which is also a
    z[i]=R-L;R--;//prefix of S.
  }else{
    int k=i-L;if(z[k]<R-i+1)z[i]=z[k];
    else{
      L=i;while(R<n&&s[R-L]==s[R])R++;
      z[i]=R-L;R--;}}}
int maxz=0,res=0;//usage
for(int i=1;i<n;i++){
  if(z[i]==n-i&&maxz>=n-i){res=n-i;break;}
  maxz=max(maxz,z[i]);}
```

7 Theory

DP Optimizations

1D-1D Optimizaton :

$$dp[i] = \min_{k<i}(dp[k] + cost[k][i])$$

If cost satifies quadrangle inequality, $Best[i] <= Best[i+1]$. Hence, for each i see for what all j>i, i could be the $Best[j]$ using binary search. Store the beginning of each segment in a vector. Whenever we consider a new value of i, perform the following to update the segments:

- While the new value of i is better than the value at the back of the vector, pop the back.
- Binary search in the current segment to find the turning point, and push this value of i together with the turning point onto the back of the vector.

Knuth's Optimization

$$dp[i][j] = \min_{i<k<j}(dp[i][k-1] + dp[k+1][j]) + cost[i][j]$$

If cost satisfies quadrangle inequality and it is monotonic, then $Best[i][j-1] <= Best[i][j] <= Best[i][j+1]$. So, build the dp over length (j=i+len) and for dp[i][j] traverse only from Best[i][j-1] to Best[i][j+1]. Reduces $O(N^3)$ to $O(N^2)$

Games

Grundy numbers. For a two-player, normal-play (last to move wins) game on a graph (V, E): $G(x) = \text{mex}(\{G(y) : (x, y) \in E\})$, where $\text{mex}(S) = \min\{n \geqslant 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

Sums of games.

- *Player chooses a game and makes a move in it.* Grundy number of a position is xor of grundy numbers of positions in summed games.
- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them.* A position is losing iff each game is in a losing position.
- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.
- *Player must move in all games, and loses if can't move in some game.* A position is losing if any of the games is in a losing position.

Misère Nim. A position with pile sizes $a_1, a_2, \ldots, a_n \geqslant 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*.

Green HackenBush Game If two players are playing a game where move allowed is to cut an edge of a (rooted) tree then for each node,set value of the node to xor of (1 + value of it's child) for all children of the node in the tree. For leaves, value is 0. See value of root to determine the winner. For a (rooted) graph, where each player can remove an edge from the component connected to the root and one unable to remove loses. Create bridge tfaree and do same as tree except val[node] = orignalVal[node] $\oplus$ (numEdges[node] & 1)

Maths

Angular bisector of angle ABC is line BD, where $D = \frac{BA}{|BA|} + \frac{BC}{|BC|}$.

Center of incircle of triangle ABC is at the intersection of angular bisectors, and is $\frac{a}{a+b+c}A + \frac{b}{a+b+c}B + \frac{c}{a+b+c}C$, where a, b, c are lengths of sides, opposite to vertices A, B, C. Radius $= \frac{2\Delta}{a+b+c}$.

Sums

$(m+1) * \sum_{k=0}^{n} k^m = (n+1)^{m+1} - \left(\sum_{r=2}^{m+1} \left(\binom{m+1}{r} \sum_{k=0}^{n} k^{m+1-r}\right)\right)$

$S_n = \sum_{k=1}^{n} [a+(k-1)d]r^{k-1} = \frac{a-[a+(n-1)d]r^n}{1-r} + \frac{dr(1-r^{n-1})}{(1-r)^2}$

Derangements. Number of permutations of $n = 0, 1, 2, \ldots$ elements without fixed points is $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$ Recurrence: $D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$. Corollary: number of permutations with exactly k fixed points is $\binom{n}{k} D_{n-k}$.

Catalan Numbers $C_n = \frac{1}{n+1}\binom{2n}{n}$. DP Recurrence : $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$ and $C_0 = 1$. It is the no of ways of arranging a bracket sequence of length n. Also equal to number of unlabelled binary trees of size n.

Stirling numbers of 1^{st} **kind**. $s_{n,k}$ is $(-1)^{n-k}$ times the number of permutations of n elements with exactly k permutation cycles. $|s_{n,k}| = |s_{n-1,k-1}| + (n-1)|s_{n-1,k}|$.

Stirling numbers of 2^{nd} **kind**. $S_{n,k}$ is the number of ways to partition a set of n elements into exactly k non-empty subsets. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = 1$.

Bell numbers. B_n is the number of partitions of n elements. $B_0, \ldots = 1, 1, 2, 5, 15, 52, 203, \ldots$ $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k = \sum_{k=1}^{n} S_{n,k}$. Bell triangle: $B_r = a_{r,1} = a_{r-1,r-1}$, $a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

Linear diophantine equation. $ax + by = c$. Let $d = \gcd(a, b)$. A solution exists iff $d|c$. If (x_0, y_0) is any solution, then all solutions are given by $(x, y) = (x_0 + \frac{b}{d}t, y_0 - \frac{a}{d}t)$, $t \in \mathbb{Z}$. To find some solution (x_0, y_0), use extended GCD to solve $ax_0 + by_0 = d = \gcd(a, b)$, and multiply its solutions by $\frac{c}{d}$. Linear diophantine equation in n variables: $a_1x_1 + \cdots + a_nx_n = c$ has solutions iff $\gcd(a_1, \ldots, a_n)|c$. To find some solution, let $b = \gcd(a_2, \ldots, a_n)$, solve $a_1x_1 + by = c$, and iterate with $a_2x_2 + \cdots = y$. Multiplicative inverse of a modulo m: x in $ax + my = 1$, or $a^{\phi(m)-1} \pmod{m}$.

Chinese Remainder Theorem. System $x \equiv a_i \pmod{m_i}$ for $i = 1, \ldots, n$, with pairwise relatively-prime m_i has a unique solution modulo $M = m_1m_2 \ldots m_n$: $x = a_1b_1\frac{M}{m_1} + \cdots + a_nb_n\frac{M}{m_n} \pmod{M}$, where b_i is modular inverse of $\frac{M}{m_i}$ modulo m_i.

System $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$ has solutions iff $a \equiv b \pmod{g}$, where $g = \gcd(m, n)$. The solution is unique modulo $L = \frac{mn}{g}$, and equals: $x \equiv a + T(b-a)m/g \equiv b + S(a-b)n/g \pmod{L}$, where S and T are integer solutions of $mT + nS = \gcd(m, n)$.

Euler's phi function. $\phi(n) = |\{m \in \mathbb{N}, m \leqslant n, \gcd(m, n) = 1\}|$.
$\phi(mn) = \frac{\phi(m)\phi(n)\gcd(m,n)}{\phi(\gcd(m,n))}$. $\phi(p^a) = p^{a-1}(p-1)$. $\sum_{d|n} \phi(d) = \sum_{d|n} \phi(\frac{n}{d}) = n$.

Euler's theorem. $a^{\phi(n)} \equiv 1 \pmod{n}$, if $\gcd(a, n) = 1$.

Wilson's theorem. p is prime iff $(p-1)! \equiv -1 \pmod{p}$.

Mobius function. $\mu(1) = 1$. $\mu(n) = 0$, if n is not squarefree. $\mu(n) = (-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. If for all $n \in N$, $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d)F(\frac{n}{d})$, and vice versa. $\phi(n) = \sum_{d|n} \mu(d)\frac{n}{d}$. $\sum_{d|n} \mu(d) = 1$.
If f is multiplicative, then $\sum_{d|n} \mu(d)f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n}(1 + f(p))$.

Primitive roots. If the order of g modulo m (min $n > 0$: $g^n \equiv 1 \pmod{m}$) is $\phi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i = \text{ind}_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod{m}$. $\text{ind}_g(a)$ has logarithm-like properties: $\text{ind}(1) = 0$, $\text{ind}(ab) = \text{ind}(a) + \text{ind}(b)$. If p is prime and a is not divisible by p, then congruence $x^n \equiv a \pmod{p}$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod{p}$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i \equiv a \pmod{p}$, $g^u \equiv x \pmod{p}$. $x^n \equiv a \pmod{p}$ iff $g^{nu} \equiv g^i \pmod{p}$ iff $nu \equiv i \pmod{p}$.)

Postage stamps/McNuggets problem. Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax+by$ $(x, y \geqslant 0)$, and the largest is $(a-1)(b-1)-1 = ab-a-b$.

Ballot Theorem. If A receives p votes and B receives q votes with $p > q$ then probability that A is strictly ahead of B at all times is given by $\frac{p-q}{p+q}$. If ties allowed : $\frac{p+1-q}{p+1}$

Leibniz formula for Determinants $det(A) = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} a_{\sigma(i),i}$. where $sgn(\sigma)$ is $+1/-1$ based on even/odd parity of number of inversions of permutation σ.

2D Recurrence using FFT For any 2D recurrence of the form $F_{n,p} = \sum_{i=0}^{k} a_i(n).F_{n-1,p-i}$. We can write it as follows use polynomial multiplication to compute the values of recurrence fast.

$$\sum_{i=0}^{kn} F_{n,i}.x^i = \prod_{i=1}^{n} \sum_{j=0}^{k} a_j(i).x^j$$

Trick for 2D FFT $\frac{P(1)+P(\delta)+...+P(\delta^{n-1})}{n}$ is sum of all indexes of polynomial P divisible by n where δ = n'th root of unity.

Fermat's two-squares theorem. Odd prime p can be represented as a sum of two squares iff $p \equiv 1 \pmod{4}$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k + 3$ occurs an even number of times in n's factorization.

Counting Primes Fast To count number of primes lesser than big n. Use following recurrence.
dp[n][j] = dp[n][j + 1] + dp[n/p_j][j] where dp[i][j] stores count of numbers lesser than equal to i having all prime divisors greater than equal to p_j. Precompute this for all i less than some small k and for others use the recurrence to compute in small time.

Graphs

Mirsky's Theorem The height of a poset is the maximum cardinality of a chain, a totally ordered subset of the given partial order.Mirsky's theorem states that, for every partially ordered set, the height also equals the minimum number of antichains (subsets in which no pair of elements are

ordered) into which the set may be partitioned. In such a partition, every two elements of the longest chain must go into two different antichains, so the number of antichains is always greater than or equal to the height.

Dilworth's Theorem An antichain in a partially ordered set is a set of elements no two of which are comparable to each other, and a chain is a set of elements every two of which are comparable. Dilworth's Theorem states that in any finite partially ordered set, the maximum number of elements in any antichain equals the minimum number of chains in any partition of the set into chains. Comparability graphs enjoy the nice property that the maximum independent set size is equal to the minimum number of cliques whose union covers all nodes (this is Dilworth's theorem). Furthermore, cliques in comparability graphs correspond to directed paths and vice-versa (once the edges are oriented as implied above). So, the problem is reduced to computing the minimum number of paths required in a directed graph to cover all nodes. This is solved via bipartite matching. Build a bipartite graph B with a copy of the nodes of the original graph on both sides. Call the original directed graph G and say it has n nodes. Add an edge from a node u on the "left" to a node v on the "right" in B if u-v is a directed edge in G. Then, the size of a minimum path cover in G (and, hence, a maximum independent set) is exactly n minus the maximum matching size in B. Minimum Disjoint Path Cover in a DAG is n - maximum matching in the corresponding bipartite graph with each vertex partitioned into 2 sets.

Konig's Thoerem In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover. Consider a bipartite graph where the vertices are partitioned into left (L) and right (R) sets. Suppose there is a maximum matching which partitions the edges into those used in the matching (E_m) and those not (E_0). Let T consist of all unmatched vertices from L, as well as all vertices reachable from those by going left-to-right along edges from E_0 and right-to-left along edges from E_m. This essentially means that for each unmatched vertex in L, we add into T all vertices that occur in a path alternating between edges from E_0 and E_m. Then $(L \setminus T) \cup (R \cap T)$ is a minimum vertex cover. Intuitively, vertices in T are added if they are in R and subtracted if they are in L to obtain the minimum vertex cover.

Matrix-tree theorem Let matrix $T = [t_{ij}]$, where t_{ij} is negative of the number of multiedges between i and j, for $i \neq j$, and $t_{ii} = \deg_i$. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T. If G is a multigraph and e is an edge of G, then the number $\tau(G)$ of spanning trees of G satisfies recurrence $\tau(G) = \tau(G - e) + \tau(G/e)$, when $G - e$ is the multigraph obtained by deleting e, and G/e is the contraction of G by e (multiple edges arising from the contraction are preserved.)

Cycle Spaces The (binary) cycle space of an undirected graph is the set of its Eulerian subgraphs. This set of subgraphs can be described algebraically as a vector space over the two-element finite field. One way of constructing a cycle basis is to form a spanning forest of the graph, and then for each edge e that does not belong to the forest, form a cycle C C_e consisting of e together with the path in the forest connecting the endpoints of e. The set of cycles C_e formed in this way are linearly independent (each one contains an edge e that does not belong to any of the other cycles) and has the correct size $mn + c$ to be a basis, so it necessarily is a basis. This is fundamental cycle basis.

Cut Spaces The family of all cut sets of an undirected graph is known as the cut space of the graph. It forms a vector space over the two-element finite field of arithmetic modulo two, with the symmetric difference of two cut sets as the vector addition operation, and is the orthogonal complement of the cycle space. To compute the basis vector for the cut space, consider any spanning tree of the graph. For every edge e in the spanning tree, remove the edge and consider the cut formed. Thus dimension of the basis vector for cut space is n-1.

Miscellaneous

- Number of perfect matchings of a bipartite graph is equal to the permanent of the adjacency matrix obtained. To check the parity of the number of perfect matchings, we can evaluate the permanent of the matrix in Z_2 which can be done easily coz in Z_2, Perm(A) = Deter(A).