tesla_protocol

International Institute of Information Technology - Hyderabad

Contents

1 0-Misc

1.1 Fast-IO

```
inline int scan(){ bool y = 0; int x = 0; char c =
   getchar_unlocked();
  while(c<'0'||c>'9'){ if(c=='-') y = 1; c = getchar_unlocked();}
  while(c>='0'&&c<='9'){ x=(x<<1)+(x<<3)+c-'0';
     c=getchar_unlocked();}
  if(y) return -x; return x;}
```

1.2 template

```
#define trace(...) {cerr << ">> ", __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...); }
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds; // OST: find_by_order, order_of_key
typedef tree<int, null_type, less<int>, rb_tree_tag,
   tree_order_statistics_node_update> ordered_set;
#pragma GCC optimize("Ofast")
optimize("unroll-loops")
target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
```

2 DP

2.1 1D-1D

```
// C[a][c]+C[b][d]<=C[a][d]+C[b][c] with a<=b<=c<=d, When
   A[i]<=A[i+1]
// A[i] is best position to calculate DP for i
// Use: set DP[1], arr.pb({N+1,1}), change <= to >= for Max
for each i:
  insert(i);
  int pos=lower_bound(arr.begin(),arr.end(),make_pair(i,0)) -
     arr.begin();
  pos = arr[pos].second; dp[i] = cost(pos, i);
void insert(int idx){
  while(true){
    if(arr.size() > 1){
      auto X = arr[arr.size() - 1], Y = arr[arr.size() - 2];
      int L = Y.second, H = N + 1, LL = idx, HH = N + 1;
      while(L < H){
        int mid = (L + H) >> 1;
        if(cost(Y.second, mid) <= cost(X.second, mid)) L = mid +
           1;
        else H = mid;
      }
      while(LL < HH){
        int mid = (LL + HH) >> 1;
        if(cost(X.second, mid) <= cost(idx, mid)) LL = mid + 1;
        else HH = mid;
      }
      if(L > LL){
        arr.pop_back(); continue;
      } else break;
    } else break;
  } auto X = arr[arr.size() - 1]; int L = idx, H = N + 1;
  while(L < H){
    int mid = (L + H) >> 1;
    if(cost(X.second, mid) <= cost(idx, mid)) L = mid + 1;
    else H = mid;
  }
  if(L > N) return;
  else{ arr[arr.size() - 1].first = L - 1; arr.push_back({N + 1,
     idx});}
}
```

2.2 Convex-Hull-Trick

```
for minimum : hull.addLine(a, b), hull.getBest(a)
for maximum : hull.addLine(-a, -b), -hull.getBest(a)

struct cht{
  struct Line{
    int a; long long b, val; double xLeft; bool type;
    Line(long long _a = 0 , long long _b = 0){
      a = _a; b = _b; xLeft = -1e16; type = 0; val = 0;
    }
    long long valueAt(int x) const{ return 1LL * a * x + b; }
    friend bool areParallel(const Line &l1, const Line &l2){
      return l1.a == l2.a; }
    friend double intersectX(const Line &l1 , const Line &l2){
      return areParallel(l1 , l2) ? 1e16 : 1.0 * (l2.b - l1.b) /
         (l1.a - l2.a);}
    bool operator < (const Line &l2) const{
      if(!l2.type) return a < l2.a;
      return xLeft > l2.val;
    }
  }; set < Line > hull;
  bool hasPrev(set < Line > :: iterator it){ return it !=
     hull.begin();}
  bool hasNext(set < Line > :: iterator it){
    return it != hull.end() && next(it) != hull.end();}
  bool irrelevant(const Line &l1 , const Line &l2 , const Line
     &l3){
    return intersectX(l1,l3) <= intersectX(l1,l2);}
  bool irrelevant(set < Line > :: iterator it){
    return hasPrev(it) z&& hasNext(it) && (irrelevant(*next(it),
       *it, *prev(it)));}
  set < Line > :: iterator updateLeftBorder(set < Line > ::
     iterator it){
    if(!hasNext(it)) return it;
    double val = intersectX(*it , *next(it));
    Line buf(*it); it = hull.erase(it);
    buf.xLeft = val; it = hull.insert(it, buf); return it;
  }
  void addLine(int a , long long b){
    Line l3 = Line(a, b); auto it = hull.lower_bound(l3);
    if(it != hull.end() && areParallel(*it , l3)){
      if(it -> b > b) it = hull.erase(it);
      else return;
    } it = hull.insert(it, l3);
    if(irrelevant(it)){ hull.erase(it); return; }
    while(hasPrev(it) && irrelevant(prev(it)))
       hull.erase(prev(it));
    while(hasNext(it) && irrelevant(next(it)))
       hull.erase(next(it));
    it = updateLeftBorder(it);
    if(hasPrev(it)) updateLeftBorder(prev(it));
    if(hasNext(it)) updateLeftBorder(next(it));
  }
  long long getBest(int x){
    Line q; q.val = x; q.type = 1; auto bestLine =
       hull.lower_bound(q);
    if(bestLine == hull.end()) return 1e16;
    return bestLine -> valueAt(x);
  }
}
```

2.3 Divide-and-Conquer

```
// can loop from optl to optr OR l to r
// CANNOT LOOP FROM L TO OPTL
void solve(int j, int l, int r, int optl, int optr){
  if(l > r) return;
  int mid = (l + r) >> 1; dp[j][mid] = INF;
  int low = optl, high = min(optr, mid), opt = optl;
  for(int k=low; k<=high; ++k){
    if(dp[j][mid] < get(j, k, mid)) dp[j][mid] = get(j, k, mid),
       opt = k;
  } solve(j, l, mid - 1, optl, opt); solve(j, mid + 1, r, opt,
     optr);
} (< for MAX) and (> for MIN)
```

2.4 IOI-Aliens-trick

```
// instead of dp[N][X][Y] calculate dp[N][X] and give some cost
   to try each Y and then check how many used. if used > Y,
   increase cost
// Applicable when f(x+1)  f(x) <= f(x)   f(x-1), f(x) =
   dp[N][A][X], i.e according to last dimension
double solve(){ //Check range for cost
  for(int i=1; i<=N; ++i){
    for(int j=0; j<=A; ++j){
      double &d = dp[i][j]; int &pick = p[i][j];
      d = dp[i - 1][j], pick = 0;
      if(j && d < dp[i - 1][j - 1] + X[i])
        d = dp[i - 1][j - 1] + X[i], pick = 1;
      if(d < dp[i - 1][j] + Y[i] - mid)
        d = dp[i - 1][j] + Y[i] - mid, pick = 2;
      if(j && d < dp[i - 1][j - 1] + X[i] + Y[i] - (X[i] * Y[i])
         - mid)
        d = dp[i-1][j-1] + X[i] + Y[i] - (X[i] * Y[i]) - mid;
           pick = 3;
    }
  }
}
int main(){
  double low = 0, high = 1;
  for(int i=0; i<50; ++i){
    mid = (low + high) / 2; solve(); int pos = A, c = 0;
    for(int i=N; i>0; --i){
      if(p[i][pos] == 1) --pos;
      else if(p[i][pos] == 2) ++c;
      else if(p[i][pos] == 3) --pos, ++c;
    }
    if(c > B) low = mid; else high = mid;
  } mid = high; solve(); printf("%0.5f\n", dp[N][A] + high * B);
}
```

2.5 Knuth-Optimisation

```
// when opt[i][j - 1] <= opt[i][j] <= opt[i + 1][j]
for(int i=1; i<=K; ++i) opt[N + 1][i] = N;
for(int j=1; j<=K; ++j){
  for(int i=N; i>0; --i){ //reverse to get opt[i + 1][j]
    dp[i][j] = 0; int low = max(1LL, opt[i][j - 1]), high =
       opt[i + 1][j];
    for(int k=low; k<=high; ++k)
      if(dp[k - 1][j - 1] + pre[k][i] > dp[i][j])
        dp[i][j] = dp[k - 1][j - 1] + pre[k][i], opt[i][j] = k;
  }
} // complexity is K * N, second N loop is amortized
```

2.6 Subset-Sum

```
// test if W weight is possible, c[i] is freq of i
// Push frequency of X to (X * 2)
int solve(int W){
  v.clear(); dp.reset();
  for(int i=1; i<=W; ++i){
    if(c[i] == 0) continue;
    if(c[i] & 1){ --c[i]; v.push_back(i); }
    else{
      --c[i]; --c[i];
      v.push_back(i); v.push_back(i);
    } int temp = (i << 1), inc = (c[i] >> 1);
    if(temp <= W) c[temp] += inc;
  }
  dp.set(0); for(auto it:v) dp |= (dp << it);
  return dp.test(W);
}
```

3 DS

3.1 Persistent

```
struct node {
        int l, r, val;
        node() {l = r = val = 0;}
};
int curr, root[N]; node seg[4 * N * LOGN];
void init() { curr = 1;
        seg[curr].l = seg[curr].r = seg[curr].val = 0; }
// create a new node.
int nnode(int val, int l, int r) {
        seg[curr].val = val, seg[curr].l = l, seg[curr].r = r;
        return curr++; }
// cur: current_node(along which are we travelling, idx: index
    to be insert at);
int Insert(int lo, int hi, int cur, int idx, int val) {
        if (idx < lo || idx > hi) return cur;
        else if (lo == hi) return nnode(val, 0, 0);
        int mid = (lo + hi) >> 1;
        int pos = nnode(-1, Insert(lo, mid, seg[cur].l, idx,
            val), Insert(mid + 1, hi, seg[cur].r, idx, val));
        seg[pos].val = max(seg[seg[pos].l].val,
            seg[seg[pos].r].val);
        return pos;
}
```

3.2 Treap

```
struct node {//combine function is refresh
  int value,priority,sz,reverse;
  node *left,*right;
}; typedef node* pnode;
int getsz(pnode t) { return t?t->sz:0; }
void refresh(pnode t) { if (!t) return;
  t->sz=getsz(t->left)+getsz(t->right)+1;
} void lazyupdate(pnode t) { if(!t || !t->reverse) return;
  swap(t->left,t->right);
  if(t->left) t->left->reverse^=1 ;
  if(t->right) t->right->reverse^=1 ;
  t->reverse=0; refresh(t);
}void merge(pnode &t,pnode l ,pnode r) {
  lazyupdate(l); lazyupdate(r);
  if(!l || !r) t=(l?l:r);
  else if(l->priority>r->priority) merge(l->right,l->right,r),
      t=l;
  else merge(r->left,l,r->left), t=r; refresh(t);
}void split(pnode t,pnode &l,pnode &r,int pos,int add) {
    lazyupdate(t);
  if(!t) { l=r=t; return; }
  int curr_pos=add+getsz(t->left)+1;
  if(curr_pos<=pos) split(t->right,t->right,r,pos,curr_pos), l=t;
  else split(t->left,l,t->left,pos,add), r=t;
  refresh(t);
} pnode create(int val) { pnode temp=new node();
  temp->priority=rand(); temp->sz=1;
  temp->left=temp->right=NULL; temp->value=val;
  temp->reverse=0; return temp;
} void insert(pnode &t,int pos,int val) { pnode l,r;
  split(t,l,r,pos-1,0); merge(l,l,create(val)); merge(t,l,r);
} void print(pnode &t) { if(!t) return;
  lazyupdate(t); print(t->left);
  cout<<t->value<<" "; print(t->right);
}
```

3.3 bit

```
void range_update(int val, int a, int b) {// range update range
    query
  update(bit1, val, a); update(bit1, -val, b+1);
  update(bit2, (a-1)*val, a); update(bit2, -b*val ,b+1); }
```

```
int pre_sum(int idx) {// now it gives sum from 1-idx.
  return idx * query(bit1, idx) - query(bit2, idx); }
```

4 flows

4.1 MCMF

```
// Set value for NODES and INF(also check in spfa function),
    call addEdge(u, v, capacity, cost)
// and for the answer use the loop in the end of main.
const int NODES = 256, INF = 600000000, INF2 = 1000000000;
int dis[NODES], prev2[NODES], inQ[NODES];
struct edge { int to, cap, cost; };
vector<edge> e; vector<int> g[NODES];
void addEdge(int u, int v, int cap, int cost) {
        g[u].push_back(e.size()); e.push_back({v, cap, cost});
        g[v].push_back(e.size()); e.push_back({u, 0, -cost});
}
int spfa(int start, int sink) {
        memset(dis, 64, sizeof(dis));
        dis[start] = 0; prev2[start] = -1; inQ[start] = 1;
        queue<int> q; q.push(start);
        while(!q.empty()) {
                int u = q.front(); q.pop(); inQ[u] = 0;
                for(auto& v: g[u])
                        if(e[v].cap && dis[e[v].to] > dis[u] +
                            e[v].cost) {
                                prev2[e[v].to] = v;
                                dis[e[v].to] = dis[u] + e[v].cost;
                                if (!inQ[e[v].to]) q.push(e[v].to);
                                inQ[e[v].to] = 1;
                        } }
        if (dis[sink] > INF) return INF;
        int temp = prev2[sink], aug;
        aug = e[temp].cap;
        while(temp!=-1) {
                aug = min(aug, e[temp].cap);
                temp = prev2[e[1^temp].to]; }
        temp = prev2[sink];
        while(temp!=-1) {
                e[temp].cap -= aug;
                e[1^temp].cap += aug;
                temp = prev2[e[1^temp].to]; }
        return (dis[sink]*aug); }
while(true) /*** in main ***/
        int temp = spfa(s, t); // min cost is returned
        if(temp == INF) break; ans += temp;
```

4.2 dinics

```
class Dinics {
public: typedef int FT; // can use float/double
        static const FT INF = 1e9; // maximum capacity
        static const FT EPS = 0; // minimum capacity/flow change
        int nodes, src, dest;
        vector<int> dist, q, work;
        struct Edge { int to, rev; FT f, cap; };
        vector< vector<Edge> > g;

        bool dinic_bfs() {
          fill(dist.begin(), dist.end(), -1);
          dist[src] = 0;
          int qt = 0;
          q[qt++] = src;
          for (int qh = 0; qh < qt; qh++) {
            int u = q[qh];
            for (int j = 0; j < (int) g[u].size(); j++) {
              Edge &e = g[u][j]; int v = e.to;
              if (dist[v] < 0 && e.f < e.cap)
                dist[v] = dist[u] + 1; q[qt++] = v;
            }
          }  return dist[dest] >= 0;
        }
        int dinic_dfs(int u, int f) {
          if (u == dest) return f;
          for (int &i = work[u]; i < (int) g[u].size(); i++) {
            Edge &e = g[u][i];
            if (e.cap <= e.f) continue;
            int v = e.to;
            if (dist[v] == dist[u] + 1) {
              FT df = dinic_dfs(v, min(f, e.cap - e.f));
              if (df > 0) { e.f += df, g[v][e.rev].f -= df;
                return df; }
            }
          } return 0;
        }

        Dinics(int n): dist(n, 0), q(n, 0), work(n, 0), g(n),
            nodes(n) {}
        // *** s->t (cap); t->s (rcap)
        void addEdge(int s, int t, FT cap, FT rcap = 0) {
          g[s].push_back({t, (int) g[t].size(), 0, cap});
          g[t].push_back({s, (int) g[s].size() - 1, 0, rcap});
        } // ***
        FT maxFlow(int _src, int _dest) {
          src = _src, dest = _dest;
          FT result = 0, delta;
          while (dinic_bfs()) {
            fill(work.begin(), work.end(), 0);
            while ((delta = dinic_dfs(src, INF)) > EPS) result +=
                delta;
          } return result;
        }
};
```

4.3 hopcroft-karp

```
class HopcroftKarp {
public: static const int INF = 1e9;
  int U, V, nil;
  vector<int> pairU, pairV, dist;
  vector< vector<int> > adj;
  bool bfs() {
    queue<int> q;
    for (int u = 0; u < U; u++)
      if (pairU[u] == nil) dist[u] = 0, q.push(u);
      else dist[u] = INF;
    dist[nil] = INF;
    while (not q.empty()) {
      int u = q.front(); q.pop();
      if (dist[u] >= dist[nil]) continue;
      for (int v: adj[u])
        if (dist[pairV[v]] == INF)
          dist[pairV[v]] = dist[u] + 1, q.push(pairV[v]);
    }  return dist[nil] != INF;
  }
  bool dfs(int u) {
    if (u == nil) return true;
    for (int v: adj[u])
      if (dist[pairV[v]] == dist[u] + 1)
        if (dfs(pairV[v])) {
          pairV[v] = u, pairU[u] = v;
          return true;
        }
    dist[u] = INF; return false;
  }
public:
  HopcroftKarp(int U_, int V_) {
    nil = U = V = max(U_, V_);
    adj.resize(U + 1); dist.resize(U + 1);
    pairU.resize(U + 1); pairV.resize(V);
  }
  void addEdge(int u, int v) { adj[u].push_back(v); }
  int maxMatch() {
    fill(pairU.begin(), pairU.end(), nil);
    fill(pairV.begin(), pairV.end(), nil);
    int res = 0;
    while (bfs()) for (int u = 0; u < U; u++)
        if (pairU[u] == nil && dfs(u)) res++;
    return res;
  }
};
```

5 geometry

5.1 Geometry3D

```
#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{
public:
  point a,n; //point on plane && vector normal to the plane
  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(db x, db 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(db x1, db y1, db z1, db x2, db y2, db z2,
    db 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(db x1, db y1, db z1,db x2, db y2, db z2, db
    px, db py, db 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;
}
```

5.2 complex

```
struct Complex { Cord x, y;
  Complex(Cord xx, Cord yy, bool isPolar = false) {
    if (isPolar) {
      x = xx * cos(yy); y = xx * sin(yy);
    } else { x = xx, y = yy; } }
  Complex(Point a, Point b) {
    x = b.x - a.x; y = b.y - a.y; }
  Cord mod() { return hypot(x, y); }
  Cord angle() {
    if (isZero(x)) return (y > 0 ? PI / 2 : -PI / 2);
    return atan(y / x); }
  void add(const Complex& b) { x += b.x; y += b.y; }
  void mult(const Complex& b) {
    Cord tx = x * b.x - y * b.y;
    Cord ty = x * b.y + y * b.x;
    x = tx, y = ty; }
  void rotate(Cord ang) { mult(Complex(1.0, ang, true)); }
  void scale(Cord len) { x *= len; y *= len; }
  void unit() { scale(1.0 / mod()); }
}; typedef Complex vect;
Cord dot(vect a, vect b) { return a.x * b.x + a.y * b.y; }
Cord cross(vect a, vect b) { return a.x * b.y - b.x * a.y; }
Cord angle(Point a, Point o, Point b) {
  auto oa = vect(o, a), ob = vect(o, b);
  oa.unit(); ob.unit(); return acos(dot(oa, ob)); }
```

5.3 convex-hull

```
vector<Point> convexHull(vector<Point> P) {
  vector<Point> up, dn;
  sort(P.begin(), P.end(), compareX);
  for (auto& p: P) {
    while (up.size() > 1 && !cw(*(up.end() - 2), *(up.end() -
        1), p))
      up.pop_back();
    while (dn.size() > 1 && !ccw(*(dn.end() - 2), *(dn.end() -
        1), p))
      dn.pop_back();
    up.push_back(p); dn.push_back(p);
  }
  for (int i = dn.size() - 2; i > 0; i--) up.push_back(dn[i]);
  return up;
}
```

5.4 geometry

```
typedef long double coord;
const coord INF = 1e18, EPS = 1e-9, PI = acos(-1);
bool zero(coord val) { return abs(val) <= EPS; }

struct Point { coord x, y; };
typedef pair<Point, Point> Segment;
// Det:- +ve = cw, -ve = ccw, 0 = collinear
coord det(Point A, Point B, Point C) {
  return (A.y - B.y) * (C.x - B.x) - (C.y - B.y) * (A.x - B.x); }
coord sgn(x) { return x == 0 ? 0 : x > 0 ? 1 : -1; }

inline coord dist2(Point a, Point b) {
  coord dx = a.x - b.x, dy = a.y - b.y;
  return (dx * dx + dy * dy);
}
inline coord dist(Point a, Point b) { return sqrt(dist(a, b)); }

Point rotate(Point p, coord ang) { // CW
  return Point(p.x*cos(ang) - p.y*sin(ang),
      p.x*sin(ang)+p.y*cos(ang));
}

class line{
public:
  double a,b,c;line(){a=b=c=0;}
  line(point p1,point p2){
    if(fabs(p1.x-p2.x)<EPS)//vertical line
    a=1.0,b=0.0,c=-p1.x;
    else
    a=-(p2.y-p1.y)/(p2.x-p1.x),b=1,c=-(p1.y + a*p1.x);
  }//slope : m and point p on line
  line(point p,double m){a=-m;b=1;c=-(a*p.x+b*p.y);}
};bool areParallel(line l1,line l2){
  return ((fabs(l1.a-l2.a)<EPS) && (fabs(l1.b-l2.b)<EPS));
}bool areSame(line l1,line l2){
  return (areParallel(l1,l2) && (fabs(l1.c-l2.c)<EPS));
}bool areIntersect(line l1,line l2,point &p){
  if(areParallel(l1,l2)) return false;
  //solve system of 2 linear algebraic equations with 2 unknowns
  p.x = (l2.b*l1.c - l1.b*l2.c)/(l2.a*l1.b - l1.a*l2.b);
  //special case for vertical line to avoid division by zero.
  if(fabs(l1.b)>EPS)p.y= -(l1.a*p.x + l1.c);
  else p.y = -(l2.a*p.x + l2.c);
  return true;
}/************* Vector Algebra *************/
```

```
class vect{
public:
  double x,y;vect():x(0),y(0){}
  vect(double _x,double _y):x(_x),y(_y){}
  vect(point a,point b):x(b.x-a.x),y(b.y-a.y){}
};double dot(vect a,vect b){return (a.x*b.x + a.y*b.y);}
double cross(vect a,vect b){return a.x*b.y - a.y*b.x;}
vect scale(vect v,double s){//Returns vector of length s along v
  double k = s/sqrt(dot(v,v));return vect(k*v.x,k*v.y);
}//Move point p along direction v by length of v
point translate(point p,vect v){return point(p.x+v.x,p.y+v.y);}
/************* Miscellaneous **************/
//dist of p from line(a,b).c:closest point to p on line l
double distToLine(point p,point a,point b,point &c){
  vect ap(a,p),ab(a,b);double u = dot(ap,ab)/sqrt(dot(ab,ab));
  c = translate(a,scale(ab,u));return dist(p,c);
}//dist of p from line-segment(a,b).c:closest point to p on
   line-segment
double distToLineSegment(point p,point a,point b,point &c){
  vect ap(a,p),ab(a,b);double u = dot(ap,ab)/dot(ab,ab);
  if(u<0.0){c=a;return dist(p,a);}//closer to a
  if(u>1.0){c=b;return dist(p,b);}//closer to b
  return distToLine(p,a,b,c);//run dist to line as above
}//returns angle aob in rad
double angle(point a,point o,point b){vect oa(o,a),ob(o,b);
  return acos(dot(oa,ob)/sqrt(dot(oa,oa)*dot(ob,ob)));
}//returns true if point r is on left side of line pq
bool ccw(point p,point q,point r){//>= to accept
  return cross(vect(p,q),vect(p,r))>0;//collinear points
}//returns true if point r is on same line as p,q
bool collinear(point p,point q,point r){
  return fabs(cross(vect(p,q),vect(p,r)))<EPS;
}/************* Polygons **************/
//Polygon is represented as vector of counter-clockwise points,
//with last point equal to first point.
double area(vector<point> &P){
  double res=0;
  for(int i=0;i<SZ(P)-1;i++)
  res+=(P[i].x*P[i+1].y-P[i+1].x*P[i].y);
  return fabs(res);
}//Check if given polygon is Convex
bool isConvex(vector<point> &P){
  if(SZ(P)<=3)return false;
  bool isLeft=ccw(P[0],P[1],P[2]);
  for(int i=1;i<SZ(P)-1;i++)
  if(ccw(P[i],P[i+1],P[(i+2)==SZ(P)?1:i+2]) != isLeft)
  return false;
  return true;
}//Returns true if point p is inside (concave/convex)P
bool inPolygon(point p,vector<point>& P){
  if(!SZ(P))return false;
  double sum=0;
  for(int i=0;i<SZ(P)-1;i++)
  if(ccw(p,P[i],P[i+1]))sum+=angle(P[i],p,P[i+1]);
  else sum-=angle(P[i],p,P[i+1]);
  return fabs(fabs(sum)-2*PI)<EPS;
}
```

6 graph

6.1 2SAT

```
/* [u => v: edge u --> v] [u or v: ~u => v, ~v => u]
[u xor v: (u or v) and (~u or ~v)]
[u eq v: (u or ~v) and (~u or v)] [u: u or u]; */
vector<int> revadj[NODES], adj[NODES], top_sort, working;
int marked[NODES], comp[NODES], c = 0;
stack<int> finishing;
void dfs1(int v) {
  marked[v] = true;
  for (auto i: adj[v]) if (marked[i] == 0) dfs1(i);
  top_sort.push_back(v);
  finishing.push(v); }
void dfs2(int v) {
  marked[v] = true;
  for (auto i: revadj[v]) if (marked[i] == 0) dfs2(i);
  comp[v] = c;
}// in main
for(int i=2; i<=(2*m+1); ++i) if(!marked[i]) dfs1(i);
memset(marked, 0, sizeof marked);
while(finishing.size()) {
  if(!marked[finishing.top()]){
    c++; dfs2(finishing.top());
  } finishing.pop();
}
for(int i=2; i<=(2*m+1); ++i)
  if(comp[i] == comp[i^1]) { printf("No\n"); return 0; }
memset(marked, 0, sizeof marked);
for(auto &z: top_sort){
  if(marked[z>>1] == 0) {
    marked[z>>1] = 1;
    if((z & 1) == 0) working.push_back(z>>1);
  } } // print working
```

6.2 Bridge-Tree

```
void dfs(int cur, int parent){
  disc[cur] = low[cur] = ++t;
  visited[cur] = true;
  for(auto it:v[cur]){
    if(visited[it] == 0){
      dfs(it, cur);
      low[cur] = min(low[cur], low[it]);
      if(MAP[{cur, it}] > 1 or MAP[{it, cur}] >
         1)arr.push_back({cur, it})
      else if(low[it] > disc[cur]);//these are bridge edges
      else arr.push_back({cur, it});
    }
    else if(it != parent){
      low[cur] = min(low[cur], disc[it]);
      arr.push_back({cur, it});
    }
  }
}
int find(int x){ return dsu[x] = (x == dsu[x]) ? x :
   find(dsu[x]); }
void join(int x, int y){ dsu[find(y)] = find(x);}
// returns a node in tree, tree is graph
int bridge_tree(){
  int root = -1;
  for(int i=1; i<=N; ++i){
    dsu[i] = i;
    if(visited[i]) continue; else dfs(i, 0);
  }
  for(auto it:arr) join(it.first, it.second);
  for(int i=1; i<=N; ++i){
    for(auto it:v[i]){
      if(find(i) != find(it)) tree[find(i)].push_back(find(it));
    } if(find(i) == i) root = i;
  } return root;
}
void dfsLCA(int cur, int parent){
  start[cur] = ++t; dp[cur][0] = parent;
  lev[cur] = lev[parent] + 1;
  for(auto it:tree[cur]){
    if(it != parent) dfsLCA(it, cur);
  } finish[cur] = ++t;
}
bool isAncestor(int u,int a){
    if(u==a) return true;
    if(start[u]<start[a] && finish[u]>finish[a]) return true;
    return false;
}
pair<int, int> getCommonPath(int u,int a,int v,int b){
    if(!isAncestor(v,a)) return make_pair(0,0);
    int x=LCA(a,b);
    if(lev[v]<lev[u]){if(isAncestor(u,x)) return make_pair(u,x);}
    else{ if(isAncestor(v,x)) return make_pair(v,x);}
    return make_pair(0,0);
}//length of path along a,b - length of common path of (a,b)
    and (c,d)
int get_answer(int a, int b, int c, int d){
    int u=LCA(a,b),v=LCA(c,d); int ret=getdist(a,b,u);
    pair<int, int> X;
    X=getCommonPath(u,a,v,c); ret -=
        getdist(X.F,X.S,LCA(X.F,X.S));
    X=getCommonPath(u,a,v,d); ret -=
        getdist(X.F,X.S,LCA(X.F,X.S));
    X=getCommonPath(u,b,v,c); ret -=
        getdist(X.F,X.S,LCA(X.F,X.S));
    X=getCommonPath(u,b,v,d); ret -=
        getdist(X.F,X.S,LCA(X.F,X.S));
    return ret;
}
```

6.3 Centroid

```
//call dfsMCT,change solve and dfs1
vi vpart[L], g[L]; //H: depth
bool vis[L];//par:parent in cent
int SZ[L], par[L], part[L], H[L], cpart;//cpart:no of parts in
    cent
//vpart[i]:nodes in part i
int dfsSZ(int v, int p = -1) {
  SZ[v] = 1;
  for(int u: g[v]) if(!vis[u] && u!=p) SZ[v]+=dfsSZ(u, v);
  return SZ[v]; }
int dfsFC(int v, int r, int p = -1) {
  for(int u: g[v]) if(!vis[u] && u!=p && SZ[u] > SZ[r] / 2)
    return dfsFC(u, r, v);
  return v; }
void dfs1(int v, int r, int p) {
  if(p == r) part[v] = ++cpart, vpart[cpart].clear();
  else if(p != -1) part[v] = part[p];
  if(p != -1) vpart[cpart].push_back(v);
  for(int u: g[v])
    if(!vis[u] && u != p) H[u] = H[v] + 1, dfs1(u, r, v); }
void solve(int r,int szr) {
  H[r] = cpart = 0;
  dfs1(r, r, -1);
  rep(i, 1, cpart + 1) { /* update all */
    for(int u: vpart[i]) /* add i part */;
  }
  rep(i, 1, cpart+1) {
     for(int u: vpart[i]) /* rem i part */;
     for(int u: vpart[i]) /* update ans */;
     for(int u: vpart[i]) /* add i part */;
  }
  /* ans for root */ ;
  // if the container used for storing isn't local.
  rep(i, 1, cpart + 1) { /* remove all */
    for(int u: vpart[i]) /* remove i part */; } }
void dfsMCT (int u, int p = -1) {
  dfsSZ(u);
  int r = dfsFC(u, u); par[r] = p;
  solve(r, SZ[u]); vis[r] = true;
  for(int u: g[r]) if(!vis[u]) dfsMCT(u,r); }
```

6.4 Dominator-tree

```
vector<int> g[N], tree[N], rg[N], bucket[N];
int sdom[N],par[N],dom[N],dsu[N],label[N];
int arr[N],rev[N],T;//1-Based directed graph input
int Find(int u,int x=0){
  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){ //Add an edge u-->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]].push_back(arr[u]);
}}
//Build Dominator tree(in main)
dfs0(1);n=T;
for(int i=n;i>=1;i--){
  for(int j=0;j<rg[i].size();j++)
    sdom[i] = min(sdom[i],sdom[Find(rg[i][j])]);
  if(i>1)bucket[sdom[i]].push_back(i);
  for(int j=0;j<bucket[i].size();j++){
    int w = bucket[i][j],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]].push_back(rev[dom[i]]);
  tree[rev[dom[i]]].push_back(rev[i]);
}//done :)
```

6.5 HLD

```
int nch = 1, narr, sub[N], pos[N], chain[N], CH[N], CP[N];
// nch: number of chains, narr: position var for pos[].
// pos: position in basearray, CH: first node of curr chain
// CP: jump node after chain finished.
void HLD(int v, int p) {
        pos[v] = ++narr, chain[v] = nch;
        int big = 0;
        for(int u: g[v]) {
                if (u == p) continue;
                else if(!big) big = u;
                else if(sub[u] > sub[big]) big = u;
        } if(big) HLD(big, v);
        for(int u: g[v]) {
                if (u == p || u == big) continue;
                ++nch, CH[nch] = u, CP[nch] = v;
        HLD(u, v); }} // query(lo, hi, pos, l, r);
int query_up(int r, int q) {
        int ans = 0, t;
        while(chain[q] != chain[r]) {
                t = chain[q];
                ans += query(1, n, 1, pos[CH[t]], pos[q]);
                q = CP[t];
        } ans += query(1, n, 1, pos[r], pos[q]);
        return ans;}
```

6.6 Stoer-Wagner

```
LL adj[N][N], wt[N]; bool pres[N], A[N]; // N = #vertices
pair<LL, VI> mincut(int n = N) {
  pair<LL, VI> res(INF, VI()); vector<VI> comp(n);
  for (int i = 0; i < n; i++) comp[i].push_back(i);
  fill(pres, pres + N, true);
  for (int ph = 0; ph < n - 1; ph++) {
    memset(A, 0, sizeof A); memset(wt, 0, sizeof wt);
    for (int i = 1, prev; i <= n - ph; i++) {
      int sel = -1;
      for (int u = 0; u < n; u++)
        if (pres[u] && !A[u] && (sel == -1 || wt[u] > wt[sel]))
          sel=u;
      if (i == n - ph) { // last phase
        if (wt[sel]<res.first) res=make_pair(wt[sel], comp[sel]);
        comp[prev].insert(comp[prev].end(), ALL(comp[sel]));
        for (int u = 0; u < n; u++)
          adj[u][prev] = adj[prev][u] += adj[u][sel];
        pres[sel] = false;
      } else {
        for (int u = 0; u < n; u++) wt[u] += adj[u][sel];
        A[sel] = true; prev = sel;
      }
    }
  }
  return res; }
```

6.7 auxillaryTree

```
namespace auxiallaryTree {
  // clear auxNodes, mark, aux;
  //auxNodes contains nodes of current aux tree
  vector <ii> g[N], aux[N]; // aux is adjList of auxtree.
  vector <int> auxNodes; // dists is distance from 1 to any node.
  int P[LOGN][N], depth[N], timer, st[N], en[N], n;
  bool rem[N];
  LL dists[N];
  bool mark[N];//mark contains marked nodes
  void dfs0 (int v, int par = 1) {
    P[0][v] = par; st[v] = timer++; //adj list is pair
    for(auto x: g[v]) {
      int u = x.fi; LL w = x.se;
      if(u == par) continue;
      depth[u] = depth[v] + 1, dists[u] = dists[v] + w;
      dfs0(u, v);
    } en[v] = timer;//*******set mark = false after all
       processing
  }
  void pre() {
    timer = 1; dfs0(1);
    rep(i, 1, LOGN) rep(j, 1, n + 1) P[i][j] = P[i - 1][P[i -
       1][j]];
  }
  bool ancestor(int par, int u) { return ((st[par] < st[u]) &&
     (en[par] >= en[u])); }
  bool cmp(int x, int y) { return st[x] < st[y]; }
  int LCA(int x, int y) {
    if(depth[x] < depth[y]) swap(x, y);
    repv(i, 0, LOGN) if(depth[P[i][x]] >= depth[y]) x = P[i][x];
    if(x == y) return x;
    repv(i, 0, LOGN) if(P[i][x] != P[i][y]) x = P[i][x], y =
       P[i][y];
    return P[0][x];
  }
  int dist(int u, int par){
    if(depth[u] < depth[par]) swap(u, par);
    return dists[u] - dists[par];
  }
  int createtree() {
    for(auto v: auxNodes) mark[v] = true, rem[v] = true;
    sort(all(auxNodes), cmp);
    int SZ = sz(auxNodes);
    rep(i, 0, SZ - 1) {
      int lc = LCA(auxNodes[i], auxNodes[i + 1]);
      if(mark[lc]) continue;
      auxNodes.push_back(lc);
      mark[lc] = true;
    }
    sort(all(auxNodes), cmp);//***call pre,clear auxNodes,create
       tree,dfs
    for(int u: auxNodes) aux[u].clear();
    stack <int> sta; sta.push(auxNodes[0]);
    rep(i, 1, sz(auxNodes)){
      int u = auxNodes[i];
      while(!ancestor(sta.top(), u)) sta.pop();
      int v = sta.top();
      LL w = dist(u, v); // edge weight of the tree
      aux[u].push_back({v, w});
      aux[v].push_back({u, w});
      sta.push(u);
    } return auxNodes[0];
  }
}
```

6.8 cutVertexTree

```
vector <int> g[N], cutVtree[N], st; //g[v]: edge-list, Take N =
   2 * required:)
int a[N], b[N], compNo[N], col[N], low[N], in[N], L[N],
   extra[N], tim, C;
bool cutV[N]; // true for cut-vertices.
int adj(int u, int i) {
       return a[i] ^ b[i] ^ u;
}
```

```
void dfs(int v) {
        int ch = 0;
        low[v] = in[v] = T++;
        for(int e: g[v]) {
                int u = adj(v, e);
                if (in[u] == -1) {
                        L[u] = L[v] + 1, st.push_back(e), dfs(u);
                        low[v] = min(low[v], low[u]);
                        if (in[v] == 0 || low[u] >= in[v]) {
                                if (in[v]||ch) cutV[v] = true;
                                ++C;
                                while(sz(st)) {
                                        int ed = st.back();
                                        col[ed] = C, st.pop_back();
                                        if (ed == e) break;
                                }
                        }
                } else if (L[u] < L[v] - 1) { // back-edge.
                        low[v] = min(low[v], in[u]),
                            st.push_back(e);
                }
                ++ch;
        }
        return;
}
void run(int n) {
        SET(in, -1); st.resize(0); C = 0;
        rep(i, 1, n + 1) if (in[i] == -1) T = 0, dfs(i);
}
void build_tree(int n) {
        run(n);
        vector < int > temp;
        SET(extra, -1);
        rep(i, 1, n + 1) {
                temp.clear();
                for(int e: g[i]) {
                        temp.push_back(col[e]);
                    }
                    sort(all(temp));
                    temp.erase(unique(all(temp)), temp.end());
                    if (temp.empty()) {
                        compNo[i] = C + i, extra[C + i] = 0;
                    } else if (sz(temp) == 1) { // belongs to only 1
                        component.
                        compNo[i] = temp[0], extra[temp[0]] = 1;
                    } else { // cutVertex
                        compNo[i] = C + i, extra[C + i] = 0;
                        for(int u: temp) {
                                extra[u] = 0;
                                cutVtree[C + i].push_back(u);
                                cutVtree[u].push_back(C + i);
                        }
                    }
                }
                return;
}
```

6.9 tarjan

```
const int N = 1e5; // nodes
int dfsno=0, pre[N], low[N], onstack[N];
int cno, comp[N]; // forms toposort: largest cno=root
vector<int> adj[N];
stack<int> stk;
void dfs(int u) {
  if (pre[u] > 0) return;
  pre[u] = low[u] = ++dfsno;
  stk.push(u); onstack[u] = true;
  for (int v: adj[u]) {
    if (pre[v] == 0) dfs(v), low[u] = min(low[u], low[v]);
    else if (onstack[v]) low[u] = min(low[u], pre[v]);
  }
  if (pre[u] == low[u]) {
    int v; do {
      v = stk.top(); stk.pop();
      onstack[v] = false; comp[v] = cno;
    } while (v != u);
    ++cno;
  }
}
// for (int i=0;i<n;i++) dfs(i);
```

7 math

7.1 FFT

```
// Make 2 vector<base> a and b with values: coefficient of x^0,
   x^1, x^2 ...
// Resize them to double the size as powers double after
   multiplication.
// Then call fft(a, 0) and fft(b, 0) to convert them to point
   form.
// Then multiply using ans[i] = a[i]*b[i].
// Then call fft(ans, 1) to get back to coefficient form.

#define base complex<double>
const double PI=acos(-1);
vector<base> a, ans; // variables
inline void fft(vector<base> &a, bool invert) {
        int logn=0, n=a.size();
        while((1<<logn)<n) ++logn;
        for(int i=1, j=0; i<n; ++i) {
                int bit = (n>>1);
                for(; j>=bit; bit>>=1) j -= bit;
                j += bit;
                if(i < j) swap (a[i], a[j]);
        }
        for(int len=2;len<=n;(len<<=1)) {
                double ang = 2*PI/len;
                if(invert) ang = -ang;
                base wlen(cos(ang), sin(ang));
                for(int i=0; i<n; i+=len) {
                        base w(1);
                        for(int j=0; j<(len/2); ++j) {
                                base u = a[i+j], v = w*a[i+j+len/2];
                                a[i+j] = u+v;
                                a[i+j+len/2] = u-v;
                                w *= wlen;
                        }
                }
        }
        if(invert) for(int i=0; i<n; ++i) a[i] /= n;
}

// This function exponentiates a polynomial to degree k.
inline void exp(int k) {
        ans.push_back(1);
        for(int i=0; k; ++i) {
                if(k&(1<<i)) {
                        ans.resize(max(ans.size(),a.size()));
                        a.resize(max(ans.size(),a.size()));
                        a.resize(a.size()<<1);
                        ans.resize(ans.size()<<1);
                        fft(a, 0); fft(ans, 0);
                        for(int i=0; i<ans.size(); ++i)
                           ans[i]*=a[i];
                        fft(ans, 1);
                        for(int i=0; i<ans.size(); ++i) {
                                if(real(ans[i])>0.5) ans[i]=1;
                                else ans[i]=0;
                        }
                        fft(a, 1);
                        k=(k^(1<<i));
                }
                else a.resize(a.size()<<1);
                fft(a, 0);
                for(int i=0;i<a.size();++i) a[i] = a[i]*a[i];
                fft(a, 1);
                for(int i=0;i<a.size();++i){
                        if(real(a[i])>0.5) a[i]=1;
                        else a[i]=0;
                }
        }
}
```

7.2 Mobius

```
vi MF(int N) {//return vector with mobious inverse of 1->N
    vi MB(N+1,0) ; ll temp ;
    MB[1] = 2 ;
    rep(i,2,N+1) if(!MB[i]) {
        MB[i] = 1 ; temp = (ll)i*(ll)i ;
        if(temp<=N) for(int j=temp ;j<=N;j+=temp) MB[j]=-1 ;
        for(int j = i<<1;j<=N;j+=i) if(MB[j]!=-1) ++MB[j] ;
    } FEN(i,N) MB[i]=(MB[i]==-1)?0:(MB[i]&1)?-1:1;
    return MB ;
}vi dirichletConv(vi &A,vi &B,int N){//H(n)=sum(x,y:x*y=n)
    A[x]*B[y]for[1,N]
    vi ans(N+1,0) ;
    FEN(i,N)for(int j=i;j<=N;j+=i)ans[j]+=A[i]*B[j/i] ;
    return ans ;
}
```

7.3 NTT

```
struct NTT{ int phi(int m){ int ans=m ;//set L=2^(max)>=max
    input size
for(int p=2;p*p<=m;++p) if(m%p==0) {while(m%p==0) m/=p ; ans -=
    ans/p ;}
 if(m>1) ans -= ans/m ; return ans ;}//set
    mod,r=g^((mod-1)/L),rinv=r^-1
const static int L=1<<19;int mod,r,rinv,w[L>>1],iw[L>>1] ; vi
    rev;
int gen(int m) {int p = phi(m) , t=p ; vi d ;for(int
    i=2;i*i<=t;++i) if(t%i==0){
while(t%i==0) t/=i; d.pb(i);} if(t!=1) d.pb(t) ; rep(i,2,m) {
    bool ans=1 ;
for(auto &it:d) if(pwmod(i,p/it,mod)==1)
    {ans=false;break;}if(ans) return i ;}}
  void prepower(){w[0]=iw[0]=1;//call prepower
rep(i,1,L>>1)
    w[i]=(w[i-1]*(ll)r)%mod,iw[i]=(iw[i-1]*(ll)rinv)%mod;}
void revb(int nl) { static int l=-1; if(nl==l) return; int
    t,j,k=0;l=nl;
rev.resize(l+1); while ((1 << k) < nl) ++k; FEN(i,l) {
    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 ntt(vi &poly,bool inv=false){int len,l,u,v; revb(sz(poly));
FN(i,sz(poly))if(rev[i]>i) swap(poly[i],poly[rev[i]]);
for(len = 2, l = 1; len<=sz(poly); len +=len, l +=l){
for(int i=0;i<sz(poly); i += len){int pi=0,add=L/len,*W
    =inv?iw:w ;
FN(j,l){ u=poly[i+j];v = ((ll)poly[i+j+l]*(ll)W[pi])%mod;
poly[i+j]=u+v<mod?u+v: u+v-mod;poly[i+j+l]=u-v>=0 ? u-v :
    u-v+mod ;
pi+=add; }}}if(inv){ int xrev = pwmod(sz(poly),mod-2,mod) ;
FN(i,sz(poly)) poly[i] = ((ll)poly[i]*(ll)xrev)%mod ;}}
vi multiply(vi &a,vi &b){int bi=1, sz1 = sz(a)+sz(b), rsz;
while((1<<bi)<sz1) ++bi; rsz = (1<<bi);
vi poly1(rsz,0) ;FN(i,sz(a)) poly1[i]=a[i] ;
vi poly2(rsz,0) ;FN(i,sz(b)) poly2[i]=b[i] ; ntt(poly1) ;
    ntt(poly2) ;
FN(i,rsz)poly1[i]=(poly1[i]*(ll)poly2[i])%mod ; ntt(poly1,1)
    ;return poly1;}} ;
```

7.4 linear-algebra

7.4.1 GaussJordan

```
// Gauss-Jordan elimination with full pivoting. (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();
  vector<int> 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;
}
```

7.4.2 GaussModP

```
// Solves systems of linear modular equations.
// 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 its value = ans[i].
// Time complexity: O(R * C^2) - MAXC is the number of columns
// * uses: modpow(a, n, m): a^n % m
typedef long long LL;
const int MAXC = 1001;int row[MAXC];LL ans[MAXC];
LL inv(LL x, LL mod) { return modpow(x, mod-2, mod); }

int run(LL mat[][MAXC], int R, int C, LL mod) {
  for (int i=0;i<C;i++) row[i] = -1;
  int r = 0;
  for (int c=0;c<(C);++c) {
    int k = r;
    while (k < R && mat[k][c] == 0) ++k;
    if(k==R)continue;
    for (int j = (0); j < (C+1); ++j) swap(mat[r][j],mat[k][j]);
    LL div=inv(mat[r][c],mod);
    for (int i = (0); i < (R); ++i) if(i!=r){
      LL w = mat[i][c]*(mod-div)%mod;
      for (int j = (0); j < (C+1); ++j) mat[i][j] = (mat[i][j] +
          mat[r][j] * w) % mod;
    }
    row[c] = r++;
  }
  for (int i=0;i<C;++i) {
    int r = row[i];
    ans[i] = r==-1 ? 0 : (mat[r][C] * inv(mat[r][i],mod)) % mod;
  }
  for (int i=r;i<R;++i) if(mat[i][C]) return -1;
  return C - r;
}
```

7.4.3 MatrixRank

```
// Reduced row echelon form via Gauss-Jordan elimination with
   partial
// pivoting.This can be used for computing the rank of a
   matrix.O(n^3)
// INPUT:  a[][] = an nxm matrix
// OUTPUT: rref[][] = an nxm matAix (stored in a[][]),
//          returns rank of a[][]
const double EPS =1e-10;
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;
}
```

7.5 miller-rabin

```
// false => composite; true => maybe prime
bool witness(LL N, int a, LL d) {
  LL x = modpow(a, d, N);
  if (x == 1 || x == N - 1) return true;
  for (; d != N - 1; d <<= 1) {
    x = (x * x) % N;
    if (x == 1) return false;
    if (x == N - 1) return true;
  } return false;
} int wit[] = {2,3,5,7,11,13,17,19,23,29,31,37}; //n<2^64
```

7.6 nCr-large

```
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;
  17
}
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);
}
```

7.7 number-theory

```
tuple<LL, LL, LL> extended_euclid(LL a, LL b) { // ax + by =
    gcd(a, b)
  LL x0 = 1, x1 = 0, y0 = 0, y1 = 1, r0 = a, r1 = b;
  while (r1 != 0) { LL q = r0 / r1, temp;
    temp = r1, r1 = r0 - q * r1, r0 = temp;
    temp = x1, x1 = x0 - q * x1, x0 = temp;
    temp = y1, y1 = y0 - q * y1, y0 = temp;
  }
  return make_tuple(x0, y0, g); // gcd(a, b) = r0
}

LL mod_inverse(LL a, LL n) { // returns -1 if mod_inv doesn't
    exist
  LL b, g;
  tie(b, IGNORE, g) = extended_euclid(a, n, g);
  return g == 1 ? ((b % n) + n) % n : -1;
}

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)
  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;
}
```

8 strings

8.1 KMP

```
const int N=250005;
int lps[N],ind[N];
void compute(string p) {//lps represents
  int i=1,j=0; lps[0]=0;//longest length
  while(i<sz(p)) {
    if(p[i]==p[j]) lps[i]=j+1,i++,j++;
    else { if(j) j=lps[j-1];
else lps[i++]=0;
    }}}
void KMP(string t, string p) {
  compute(p); if(sz(p)>sz(t)) return; int i=0,j=0;
  while(i<sz(t)) {//max matching string till i
    if(t[i]==p[j]) ind[i]=j+1,i++,j++;
    if(j==sz(p)) j=lps[j-1];
    else if(i<sz(t) && t[i]!=p[j]) {
  if(!j) ind[i]=0,i++; else j=lps[j-1];
}}}
```

8.2 SuffixArray

```
const int N = (int)2e5 + 10;
const int LOGN = 22;
//SA[i] =  ith  Lexicographically smallest suffixs index.
int
   RA[LOGN][N],SA[N],tempSA[N],cnt[N],msb[N],LCP[LOGN][N],dollar[N];
void cSort(int l,int k,int n) {
  SET(cnt,0);
  rep(i,0,n) cnt[(i+k<n?RA[l][i+k]:0)]++;
  int maxi=max(300,n),t;
  for(int i=0,sum=0;i<maxi;i++) {
    t = cnt[i], cnt[i] = sum, sum += t;// index
  }
  rep(i,0,n)tempSA[cnt[(SA[i]+k<n?RA[l][SA[i]+k]:0)]++]=SA[i];
  rep(i,0,n)SA[i]=tempSA[i];}//dollar[i]: next '$' in string.
void build_SA(string &s){
  int n = sz(s);
  rep(i,0,n) RA[0][i] = s[i], SA[i] = i;
  rep(i,0,LOGN-1){
    int k=(1<<i);
    if(k>=n)break;
    cSort(i,k,n);cSort(i,0,n);
    int rank=0;RA[i+1][SA[0]]=rank;
    rep(j,1,n){
      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;
  rep(i,0,N) {
    if(i>=(1<<(mx+1)))
      ++mx;msb[i]=mx;
}}
void build_LCP(string& s){
  int n=sz(s);
  rep(i,0,n-1){
    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[0][n-1]=10*N;
  rep(i,1,LOGN){
    int add=(1<<(i-1));
    if(add>=n) break;
    rep(j,0,n){
      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){
  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){//usage:
    build_msb(),build_SA(s),
  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];}//build_LCP(s)
```

8.3 aho-corasick

```
struct ahocorasick {//SZ:no of nodes
  vector <int> sufflink,out;
  vector< map<char, int> > trie;//call findnextstate
  ahocorasick() {
    out.resize(1); trie.resize(1);
  }
  inline void insert(string &s) {
    int curr = 0;//clear to reinit
    rep(i,0,sz(s)) {
      if(!trie[curr].count(s[i])) {
        trie[curr][s[i]] = sz(trie);
        trie.push_back(map<char,int>());
        out.push_back(0);
      } curr = trie[curr][s[i]];
    } ++out[curr];
  }
  inline void build_automation() {
    sufflink.resize(sz(trie));
    queue <int> q;
    for(auto x: trie[0]) {
      sufflink[x.se]=0;
      q.push(x.se);
    }
    while(!q.empty()) {
      int curr = q.front(); q.pop();
      for(auto x:trie[curr]) {
        q.push(x.se); int tmp=sufflink[curr];
        while(!trie[tmp].count(x.fi) && tmp) tmp = sufflink[tmp];
        if(trie[tmp].count(x.fi)) sufflink[x.se]=trie[tmp][x.fi];
        else sufflink[x.se]=0;
        out[x.se]+=out[sufflink[x.se]];
      }
    }
  }
  int findNextState(int curr,char ch) {
    while(curr && !trie[curr].count(ch)) curr=sufflink[curr];
    return (!trie[curr].count(ch)) ? 0 :trie[curr][ch];
  }
  int query(string &s){
    int ans=0; int curr = 0;
    rep(i,0,sz(s)) {
      curr=findNextState(curr,s[i]);
      ans+=out[curr];
    } return ans;
  }
  void clear() {
    trie.clear(); sufflink.clear(); out.clear();
    out.resize(1); trie.resize(1);
}};
```

8.4 manachar

```
int P[2*N];// P[i]: longest-len palin keeping i as center.if
string new_str(string& s) {// new_str()[i]='#'; even-length
    palin.
  string NEW;NEW+="@";NEW+="#";for(auto x:s)NEW+=x,NEW+="#";
  NEW+="$";return NEW;}//1)curr ind in right boundary, init with
int lps(string& s) { // mirror val of curr ind in actual string.
  string Q=new_str(s); int c=0,r=0;
  rep(i,1,sz(Q)-1) {//2) expand beyond current length (if
      possible)
/*1*/if (i<r)P[i]=min(r-i,P[2*c-i]); //3) Update c, r
/*2*/while(Q[i+P[i]+1]==Q[i-P[i]-1])++P[i];// if the current
/*3*/if (i+P[i]>r)c=i,r=i+P[i];}//pal expand beyond right bound
  int ml=0,palc=0;rep(i,1,sz(Q)-1)if(P[i]>ml)ml=P[i],palc=i;
return ml;}// pal=s.substr((palc-1-ml)/2,ml);palc=palc/2-1;
```

8.5 suffix-automaton

```
// To check substring/LCS, run the string on the automaton.
    Each path in the automaton is a substring(if it ends in a
    terminal node, it is a suffix)
// To find occurences of a string, run it on the automaton, and
    the number of its occurences would be number of ways to
    reach a terminal node.
// Or, we can keep reverse edges of suffix links(all prefixes
    for that substring), and number of ways to reach a root,
    would be the answer(can be used to print all answers)
struct SuffixAutomaton {
  vector<map<char,int>> edges;
  vector<int> link, length; //length[i]: longest string in i-th
      class
  int last; // index of equivalence class of whole string
  SuffixAutomaton(string s) {
    edges.push_back(map<char,int>());
    link.push_back(-1); length.push_back(0);
    last = 0;
    for(int i=0;i<s.size();i++) {
      edges.push_back(map<char,int>());
      length.push_back(i+1);
      link.push_back(0);
      int r = edges.size() - 1;
      int p = last;
      while(p >= 0 && edges[p].find(s[i]) == edges[p].end())
        edges[p][s[i]] = r, p = link[p];
      if(p != -1) {
        int q = edges[p][s[i]];
        if(length[p] + 1 == length[q]) link[r] = q;
        else {
          edges.push_back(edges[q]);
          length.push_back(length[p] + 1);
          link.push_back(link[q]);
          int qq = edges.size()-1;
          link[q] = qq; link[r] = qq;
          while(p >= 0 && edges[p][s[i]] == q)
            edges[p][s[i]] = qq, p = link[p];
        }
      }
      last = r;
    }
    vector<int> terminals; int p = last;
    while(p > 0) terminals.push_back(p), p = link[p];
  }
};
```

8.6 z-algo

```
string s; int z[2000010];
void computeZ() {
        int len = s.length(), l = -1, r = -1; z[0] = len;
        for (int i=1; i<len; ++i) {
                z[i]=0;
                if (r > i) z[i] = min(z[i-l],r-i+1);
                while (s[i+z[i]] == s[z[i]]) ++z[i];
                if (i+z[i]-1 > r) r = i+z[i]-1, l = i;
        }
}
```

9 theory

Games

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-tree and do same as tree except val[node] = orignalVal[node] $\oplus$ (numEdges[node] & 1)
StairCase Nim Stair Case from {1...N}. Ans = xor of a_{even}
k-Nim. One Player can reduce the size of {1...k} piles. Starting position is loosing iff for all $j\epsilon\{0...LOGMAX\}\ \sum_{i=1}^{N} a_i\&2^j = 0$ mod (k+1).

Maths

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 $C_0 = 1$ .No. of balanced paranthesis. No. of paths to go from one end of matrix to another but only below main diagonal. Number of full binary trees with N+1 leaves. Number of non-isomorphic ordered trees with n vertices. Number of triangulations of polygon with N+2 sides. Number of ways to tile a stairstep shape of height n with n rectangles. The number of rooted binary trees with n internal nodes. Number of standard Young tableaux whose diagram is a 2-by-n rectangle. i.e. the number of ways the numbers 1, 2, ..., 2n can be arranged in a 2-by-n rectangle so that each row and each column is increasing.

Pascal Triangle Properties:
$\sum_{i=1}^{n}\binom{i}{j} = \binom{n+1}{j+1}, \sum_{i=0}^{n/2}\binom{n-i}{i} = F_{n+1}$
$\sum_{j=0}^{k}\binom{m}{j} * \binom{n-m}{k-j} = \binom{n}{k}, \sum_{m=0}^{n}\binom{m}{j} * \binom{n-m}{k-j} = \binom{n+1}{k+1}$
$\sum_{k=-a}^{a}(-1)^k\binom{2a}{k+a} = \frac{(3a)!}{(a!)^3}, \sum_{k=-a}^{a}(-1)^k\binom{a+b}{a+k}\binom{b+c}{b+k}\binom{c+a}{c+k} = \frac{(a+b+c)!}{(a!)(b!)(c!)}$

Stirling numbers of 1st 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. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = S_{0,0} = 1$. $S_{n,0} = S_{0,n} = 0$.
$n^k = \sum_{i=0}^{n} \binom{n}{i} S_{k,i} * i!$ Take binomial transform: $S_{n,k} = \sum_{i=0}^{k} \frac{(-1)^{k-i} i^n}{i! * (k-i)!}$ This is polynomial multiplication.
Binomial Transform: $a_n = \sum_{i=0}^{n} \binom{n}{i} b_i$
$b_n = \sum_{i=0}^{n} \binom{n}{i} (-1)^(n-i) a_i$

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: $B_r = a_{r,1} = a_{r-1,r-1}$, $a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

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 $\sum_{d|n} \mu(d) f(d) = \prod_{p|n} (1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n} (1 + f(p))$. Let the problem be to find G=$\sum_{i=1}^{n} \sum_{j=i+1}^{n}$h(gcd(i,j)),here h(n) should be a multiplicative function. Re-write the equation like this: G=$\sum_{g=1}^{n}$ h(g)*cnt[g], where cnt[g] is no. of pairs such that gcd(i,j)=g. Find function f(n) such that h(n)=$\sum_{d|n}$ f(d). this can be done using mobius inversion and sieve. G=$\sum_{d=1}^{n}$ h(d)*cnt2[d], where cnt2[d] is no. of pairs such that gcd(i,j) is a multiple of d.

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 Max length chain is equal to min partitioning into antichains. Max chain is height of poset.
Dilworth's Theorem Min partition into chains is equal to max length antichain. From poset create bipartite graph. Any edge from v_i - v_j implies LV_i - RV_j. Let A be the set of vertices such that neither LV_i nor RV_i are in vertex cover. A is an antichain of size n-max matching. To get min partition into chains, take a vertex from left side, keep taking vertices till a matching exist. Consider this as a chain. Its size is n - max matching.
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.)