Heuristics, IIIT Hyderabad
Team Reference Material

December 18th, 2015.

Template

```
#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

//FILE *fin = freopen("in","r",stdin);
//FILE *fout = freopen("out","w",stdout);

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>

using namespace __gnu_pbds; //Old: pb_ds

typedef tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update>ordered_set;
// Gives find_by_order(int k) (gives kth largest (0-based)) and
// order_of_key(int) (gives num items strictly smaller)
```

.vimrc

```
===================================
set shiftwidth=3
set cindent
autocmd BufNewFile *.cpp 0r ~/start.cpp
set wrapscan
===================================
```

Graph Theory

Bi-Connected Components

```
int exist[MAXN],label[MAXN],isart[MAXN],discTime[MAXN], upTime[MAXN];
//upTime stores the smallest dfs number that can be reached from here.
//exist[i] = 1 if ith vertex exists. label[i] = component of ith vertex.
vector<int> artPoints;
vector<vector<int> > a;
stack<PII> st;
vector<vector<PII> > comps; //stores the edges belonging to each component.
int setnum;     //stores the number of sets
int dfstime;
void dfs(int cur,int par){
    discTime[cur] = ++ dfstime;
    upTime[cur] = dfstime;
    int i;
    for(i=0;i<a[cur].size();i++){
        if( a[cur][i] == par )
            continue;
      if(discTime[a[cur][i]] == -1){
            st.push(MP(cur,a[cur][i]));
            dfs(a[cur][i],cur);
            upTime[cur] = min(upTime[cur],upTime[a[cur][i]]);
            if(upTime[a[cur][i]] >= discTime[cur]){
            // a[cur][i] is a seperate component, cur is an articulation point
                while(1){
                   comps[setnum].PB(st.top());
                   label[st.top().first] = label[st.top().second] = setnum;
                   if(st.top() == MP(cur,a[cur][i])){
                        st.pop();
                        break;
                   }
                   st.pop();
                }
                ++ setnum;
                ++ isart[cur];
            }
        }
        else{
            if(discTime[cur] > discTime[a[cur][i]])  st.push(MP(cur,a[cur][i]));
            upTime[cur] = min(upTime[cur],discTime[a[cur][i]]);
        }
    }
    if( (isart[cur] && par != cur) || (par == cur && isart[cur]>1) )
        artPoints.PB(cur);
    if(par == cur && isart[cur] == 1)
        isart[cur] = 0;
    return;
}
void bicon_decomp(int n){
        dfstime = 0;
        while(!st.empty())st.pop();
        memset(discTime,-1,sizeof(discTime));
        memset(isart,0,sizeof(isart));
        for(int i=0;i<n;i++)
            if(discTime[i] == -1 && exist[i])
                dfs(i,i);
}
```

Max Flow/Min Cut - Push Relabel

```
// Running time: O(|V|^3) (~1e4 V, ~1e6 E)
// INPUT: - graph, constructed using AddEdge() - source - sink
// OUTPUT: - maximum flow value - To obtain the actual flow values, look at all edges with
// capacity > 0 (zero capacity edges are residual edges).
struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};
struct PushRelabel {
  int N; vector<vector<Edge> > G; vector<LL> excess; vector<int> dist,active,count; queue<int> Q;

  PushRelabel(int N) : N(N), G(N), excess(N), dist(N), active(N), count(2*N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));}
  void Enqueue(int v) {
    if (!active[v] && excess[v] > 0) { active[v] = true; Q.push(v); } }
  void Push(Edge &e) {
    int amt = int(min(excess[e.from], LL(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);}
  void Gap(int k) {
    for (int v = 0; v < N; v++) {
      if (dist[v] < k) continue;
      count[dist[v]]--;
      dist[v] = max(dist[v], N+1);
      count[dist[v]]++;
      Enqueue(v);}}
  void Relabel(int v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (int i = 0; i < G[v].size(); i++)
      if (G[v][i].cap - G[v][i].flow > 0)
  dist[v] = min(dist[v], dist[G[v][i].to] + 1);
    count[dist[v]]++;
    Enqueue(v);}
  void Discharge(int v) {
    for (int i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[v][i]);
    if (excess[v] > 0) {
      if (count[dist[v]] == 1)
  Gap(dist[v]);
      else
  Relabel(v);}}
  LL GetMaxFlow(int s, int t) {
    count[0] = N-1;
    count[N] = 1;
    dist[s] = N;
    active[s] = active[t] = true;
    for (int i = 0; i < G[s].size(); i++) {
      excess[s] += G[s][i].cap;
      Push(G[s][i]);
    }
    while (!Q.empty()) {
      int v = Q.front();
      Q.pop();
      active[v] = false;
      Discharge(v);
    }
    LL totflow = 0;
    for (int i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;
    return totflow;
  }};
```

Maximum Flow - Dinics.

```
class FlowDinics {
   private:
      struct edge {
         int to, back, cap;
         edge() {to = back = cap = -1;}
         edge(int _to, int _cap, int _back) : to(_to), cap(_cap), back(_back) {}
      };
      vector<vector<edge> > graph;
      VI Layers,now;
      int n, source, sink;
   public:
      FlowDinics(int _n,int _source,int _sink) : n(_n), source(_source), sink(_sink) {
         graph.resize(n);Layers.resize(n);now.resize(n);
         REP(i,n) graph[i].clear();}
      ~FlowDinics() {REP(i,n) graph[i].clear();}
      void insert(int i,int j,int cap) {
         graph[i].PB(edge(j,cap,graph[j].size()));
         graph[j].PB(edge(i,0,graph[i].size()-1));}
      int bfs() {
         fill(ALL(Layers),-1);
         Layers[sink] = 0;
         queue<int> Q; Q.push(sink);
         int top, from, to, cap, index;
         while(!Q.empty()) {
            top = Q.front(); Q.pop();
            REP(i,graph[top].size()) {
               from = graph[top][i].to; index = graph[top][i].back;
               if(graph[from][index].cap && Layers[from] == -1) {
                  Layers[from] = Layers[top] + 1;
                  Q.push(from);}}}
         return (Layers[source] != -1);}
      int dfs(int node,int val) {
         if(node == sink) {
            return val;}
         int flow, to, cap;
         for(int& start = now[node] ; start < graph[node].size(); ++start) {
            to  = graph[node][start].to;
            cap = graph[node][start].cap;
            if(cap > 0 && Layers[node] == Layers[to] + 1) {
               flow = dfs(to,min(val,cap));
               if(flow) {
                  graph[node][start].cap -= flow;
                  graph[to][graph[node][start].back].cap += flow; return flow;}}}
         return 0;}
      int flow() {
         int ret = 0, flow = 0;
         while(bfs()) {
            fill(ALL(now),0);
            while(flow = dfs(source,INF)) ret += flow;}
         return ret;}};
```

Stable Marriage

```
/* Stable Marriage Problem, O(m^2), m men and n women, n>=m, preference
 * No two couples M1W1 and M2W2 will be unstable.
 * Two couples are unstable if M1 prefers W2 over W1 & W1 prefers M2 over M1.
 * IN: L[i][]: the list of women in order of decreasing preference of man i.
 * IN: R[j][i]: the attractiveness of i to j.
 * OUT: L2R[]: mate of man i (always between 0 and n-1)
 * OUT: R2L[]: mate of woman j (or -1 if single) */
void stableMarriage(){
    int p[128] = {};
    memset( R2L, -1, sizeof( R2L ) );
    // Each man proposes...
    for( int i = 0; i < m; i++ ){
        int man = i;
        while( man >= 0 ){
        int wom;while( 1 ){wom = L[man][p[man]++];
        if( R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]] ) break;}
int hubby = R2L[wom];R2L[L2R[man] = wom] = man;man = hubby;}}}
```

Bipartite Matching

```
//for maximum bipartite matching (E * V^0.5)
#define REP(i,n) for(int i =0; i<n; i++)
const int MAXN = 10000;
struct HopcroftKarp{
 vector<int> g[MAXN];
 int n,distX[MAXN],distY[MAXN],mateX[MAXN],mateY[MAXN];
 queue<int> q;
 inline void reset(int m){ n=m; REP(i,n) g[i].clear();
 memset(mateX,-1,n*sizeof(int)); memset(mateY,-1,n*sizeof(int)); }
 inline void add_edge(int u,int v){ g[u].push_back(v); }
 inline bool bfs(){
  bool found=false; REP(i,n) if(mateX[i]==-1) q.push(i); REP(i,n) distX[i]=distY[i]=0;
  while(!q.empty()){
    int u=q.front(); q.pop();
    for(int i=0;i<(int)(g[u].size());i++){
     if(distY[g[u][i]]==0){
      distY[g[u][i]]=distX[u]+1;
      if(mateY[g[u][i]]==-1) found=true;
      else distX[mateY[g[u][i]]]=distY[g[u][i]]+1,q.push(mateY[g[u][i]]);}}}return found;}
 bool dfs(int u){
  for(int i=0;i<(int)(g[u].size());i++){
    if(distY[g[u][i]]==distX[u]+1){
     distY[g[u][i]]=0;
      if(mateY[g[u][i]]==-1||dfs(mateY[g[u][i]]))
       { mateX[u]=g[u][i]; mateY[g[u][i]]=u; return true; }}}return false;}
 int find(){
  int res=0; while(bfs()) REP(i,n) if(mateX[i]==-1) if(dfs(i)) res++; return res;}};
```

MIS in comparability graphs
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.
Vertex Cover from a bipartite matching: 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. Thus, the HopcroftKarp algorithm for finding maximum matchings in bipartite graphs may also be used to solve the vertex cover problem efficiently in these graphs.
Assignment Problem - KuhnMunkras Algorithm. Optimal Assignment.

```
// Reverse edges are not to be given any cost
// Left vertices forcibly matched, Right vertices MAY remain unmatched
// !!! nx <= ny
#include <cstring>
#include <algorithm>
using namespace std;
const int maxNode =505;
int cost[maxNode][maxNode],X[maxNode],Y[maxNode];
int hungarian(int nx, int ny){
    int Lx[maxNode], Ly[maxNode], Q[maxNode], prev[maxNode];
    memset(X, -1, sizeof X); memset(Y, -1, sizeof Y);
    memset(Lx, 0, sizeof Lx); memset(Ly, 0, sizeof Ly);
    for(int i = 0; i < nx; i++)
        for(int j = 0; j < ny; j++)
            Lx[i] = max( Lx[i], cost[i][j]);
    for(int i = 0;i < nx; ){
        memset( prev, -1, sizeof prev);
        int head = 0, tail = 0;
        for(Q[tail++] = i; head < tail && X[i] < 0; head ++){
            int u = Q[head];
            for(int v = 0; v < ny && X[i] < 0; v++){
                if( prev[v] >= 0 || Lx[u] + Ly[v] > cost[u][v] ) continue;
                if( Y[v] >= 0)
                    prev[v] = u, Q[tail++] = Y[v];
                else{
                    prev[v] = u;
                    for(int at = v; at >=0; ){
                        u = Y[at] = prev[at];
                        swap(X[u],at);
                    }
                }
            }
        }
        if( X[i] >=0) i++;
        else{
            int alpha = INF;
            for(int head = 0; head < tail; head++){
                int u = Q[head];
                for(int v = 0; v < ny; v++)
```

```
                    if( prev[v] == -1)
                        alpha = min ( alpha, Lx[u] + Ly[v] - cost[u][v] );
            }
            for(int head = 0; head < tail; head ++)
                Lx[Q[head]] -= alpha;
            for(int v = 0; v < ny; v++)
                if( prev[v] >= 0 )
                    Ly[v] += alpha;
        }
    }
    int Ans = 0;
    for (int i = 0; i < ind; i ++) if (X[i] >= 0) Ans += C[i][X[i]];
    return Ans;
}
```

MinCostMaxFlow - Bellman Ford, Works for Negative Cost Edges also.

```
#define MAXN 175
int cap[MAXN][MAXN], flow[MAXN][MAXN], cost[MAXN][MAXN], dist[MAXN], prev[MAXN];
int n, sink, source, total_cost;
bool reach[MAXN];
inline int cf(int i, int j) {
   return cap[i][j];
}
inline int costf(int i, int j) {
   return cost[i][j];
}
inline void pushf(int i, int j, int x) {
   total_cost += costf(i, j) * x;
   cap[i][j] -= x; cap[j][i] += x;
}
int augment(void) {
   bool changed;
   memset(reach,0,sizeof reach);
   reach[source] = changed = true;
   dist[source] = 0;
   while (changed) {
      changed = false;
      REP(i,n) if (reach[i]) REP(j,n) if (cf(i, j))
         if (!reach[j] || dist[i] + costf(i, j) < dist[j]) {
            reach[j] = changed = true;
            dist[j] = dist[i] + costf(i, j);
            prev[j] = i;
         }
   }
   if (!reach[sink]) return 0;
   int ret = INF;
   for (int i = sink; i != source; i = prev[i]) ret = min(ret, cap[prev[i]][i]); //pushf(prev[i], i, 1);
   for (int i = sink; i != source; i = prev[i]) pushf(prev[i], i, ret);
   return ret;
}
int mcmf(void) {
   int ret, x;
   ret = total_cost = 0;
   while (x = augment()) ret += x;
   return ret;
}
```

Min-cost max-flow —— Dijkstra

```
struct MCMF {
    typedef int ctype;
    enum { MAXN = 1000, INF = INT_MAX };
    struct Edge { int x, y; ctype cap, cost; };
    vector<Edge> E;      vector<int> adj[MAXN];
    int N, prev[MAXN];   ctype dist[MAXN], phi[MAXN];

    MCMF(int NN) : N(NN) {}
    void add(int x, int y, ctype cap, ctype cost) {  // cost >= 0
        Edge e1={x,y,cap,cost}, e2={y,x,0,-cost};
        adj[e1.x].push_back(E.size()); E.push_back(e1);
        adj[e2.x].push_back(E.size()); E.push_back(e2);
    }

    void mcmf(int s, int t, ctype &flowVal, ctype &flowCost) {
        int x;
        flowVal = flowCost = 0;  memset(phi, 0, sizeof(phi));
        while (true) {
            for (x = 0; x < N; x++) prev[x] = -1;
            for (x = 0; x < N; x++) dist[x] = INF;
            dist[s] = prev[s] = 0;

            set< pair<ctype, int> > Q;
            Q.insert(make_pair(dist[s], s));
            while (!Q.empty()) {
                x = Q.begin()->second; Q.erase(Q.begin());
                FOREACH(it, adj[x]) {
                    const Edge &e = E[*it];
                    if (e.cap <= 0) continue;
                    ctype cc = e.cost + phi[x] - phi[e.y];                    // ***
                    if (dist[x] + cc < dist[e.y]) {
                        Q.erase(make_pair(dist[e.y], e.y));
                        dist[e.y] = dist[x] + cc;
                        prev[e.y] = *it;
                        Q.insert(make_pair(dist[e.y], e.y));
                    }
                }
            }
            if (prev[t] == -1) break;

            ctype z = INF;
            for (x = t; x != s; x = E[prev[x]].x) z = min(z, E[prev[x]].cap);
            for (x = t; x != s; x = E[prev[x]].x)
                { E[prev[x]].cap -= z; E[prev[x]^1].cap += z; }
            flowVal += z;
            flowCost += z * (dist[t] - phi[s] + phi[t]);
            for (x = 0; x < N; x++) if (prev[x] != -1) phi[x] += dist[x];     // ***
        }
    }
};
```

MinCostMaxFlow - CycleCancellingAlgorithm

```
1. Repeatedly find negative cost cycles in the graph, using Bellman Ford.
2. Circulate flow across these cycles. Update Graph.
3. Algorithm ends when no cycle is found.
```

Minimum Cut - Stoer Wagner

```
 /**Returns the minimum set of edges that, when removed, disconnect the graph.
 * graph contains the adjacency matrix. graph[i][j] is 0 if there is no edge between i and j.
 * Returns 0 is the graph is disconnected.  * Complexity: O(n^3).
 **/
#define MAXN 501
#define MAXW 10000000000LL
int graph[MAXN][MAXN], v[MAXN], na[MAXN];
LL w[MAXN];
bool a[MAXN];
LL minCut(int n) {
    if(!isConnected(n)) {
        return 0;
    }
    REP (i,n) v[i] = i;
    LL best = MAXW * n * n;
    while (n > 1) {
        a[v[0]] = true;
        FOR (i,1,n) {
            a[v[i]] = false;
            na[i - 1] = i;
            w[i] = graph[v[0]][v[i]];
        }
        int prev = v[0];
        FOR (i,1,n) {
            int zj = -1;
            FOR (j,1,n) if (!a[v[j]] && (zj < 0 || w[j] > w[zj])) {
                    zj = j;
            }
            a[v[zj]] = true;
            if (i == n - 1) {
                best = min(best, w[zj]);
                REP (j,n) {
                    graph[v[j]][prev] = graph[prev][v[j]] += graph[v[zj]][v[j]];
                }
                v[zj] = v[--n];
                break;
            }
            prev = v[zj];
            FOR (j,1,n) if (!a[v[j]]) {
                w[j] += graph[v[zj]][v[j]];
            }
        }
    }
    return best;
}
```

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.)

Euler tours. Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected. Construction:

```
doit(u):
  for each edge e = (u, v) in E, do: erase e, doit(v)
  prepend u to the list of vertices in the tour
```

Stable marriages problem. While there is a free man m: let w be the most-preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

2-SAT. Build an implication graph with 2 vertices for each variable – for the variable and its inverse; for each clause $x \vee y$ add edges $(\overline{x}, y)$ and $(\overline{y}, x)$. The formula is satisfiable iff x and $\overline{x}$ are in distinct SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (i.e. Kosaraju's last step), assigning 'true' to all variables of the current SCC (if it hasn't been previously assigned 'false'), and 'false' to all inverses. (if $comp[u] > comp[notu]$, then u, else not u)

Convex Hull Trick

```
struct line{
    long long y0;
    int m;
    line(){}
    line(long long _y0, int _m):
        y0(_y0), m(_m){}
};
bool check(line a, line b, line c){
    return (a.y0 - b.y0) * (c.m - a.m)
          < (a.y0 - c.y0) * (b.m - a.m);
}
int nh,pos;
line H[MAXN];
void update(line l){
    while(nh >= 2 && !check(H[nh - 2],H[nh - 1],l)){
        if(pos == nh - 1) --pos;
        --nh;
    }
    H[nh++] = l;
}
long long eval(int id, int x){
    return H[id].y0 + (long long)H[id].m * x;
}
long long query(int x){
    while(pos + 1 < nh &&
        eval(pos,x) > eval(pos + 1,x)) ++pos;
    return eval(pos,x);
}
```

Combinatorics

Sums

$\sum_{k=0}^{n} k = n(n+1)/2$

$\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$

$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$

$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$

$\sum_{k=0}^{n} k^4 = (6n^5+15n^4+10n^3-n)/30$

$\sum_{k=0}^{n} k^5 = (2n^6+6n^5+5n^4-n^2)/12$

$\sum_{k=0}^{n} x^k = (x^{n+1}-1)/(x-1)$

$\sum_{k=0}^{n} kx^k = (x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$

$S(n,m) = \sum_{j=1}^{n} j^m = \sum_{k=1}^{m+1} F(m,k) * n^k, F(i,j) = (i/j) * F(i-1,j-1), \sum_{k=1}^{m+1} F(m,k) = 1$

$1 + x + x^2 + \cdots = 1/(1-x)$

Binomial coefficients

$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$

Number of ways to pick a multiset of size k from n elements: $\binom{n+k-1}{k}$

Number of n-tuples of non-negative integers with sum s: $\binom{s+n-1}{n-1}$, at most s: $\binom{s+n}{n}$

Number of n-tuples of positive integers with sum s: $\binom{s-1}{n-1}$

Number of lattice paths from $(0,0)$ to (a,b), restricted to east and north steps: $\binom{a+b}{a}$

Harmonic Progression.

$\sum_{k=1}^{n} 1/k = ln(n) + 0.577215664901532 + 1/(2n) - 1/(12n^2) + 1/(120n^4) \ldots$

$\binom{n}{r}$ **modulo p.**

$\binom{n}{r}$ mod p equals the product of $\binom{n_i}{r_i}$ mod p where $n_1 n_2 \ldots$ and $r_1 r_2 \ldots$ are the digits of n and r when written in base p. (e.g - $\binom{21}{3}$ mod 17 equals $\binom{1}{0} * \binom{4}{3}$ mod 17.)

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 $\binom{n}{k} D_{n-k}$.

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. $|s_{n,k}| = |s_{n-1,k-1}| + (n-1)|s_{n-1,k}|$. $\sum_{k=0}^{n} s_{n,k}\, x^k = x^{\underline{n}}$

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. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = 1$. $x^n = \sum_{k=0}^{n} S_{n,k}\, x^{\underline{k}}$

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

Number Theory

```
LL coeffA, coeffB, G;
void extGcd(LL a, LL b)
{
    if( b == 0 )
    {
        G = a;
        coeffA = 1; coeffB = 0;
    }
    else
    {
        extGcd(b, a % b);
        coeffA -= coeffB * (a/b);
        swap(coeffA, coeffB);
    }
}
```

```
/* If this system { x = r1 mod m1 ; x = r2 mod m2 }
 * has a solution, this methods returns the
 * smallest non negative solution.
 * If there is no solution, -1 is returned.*/
LL congruence( LL r1, LL m1, LL r2, LL m2)
{
    extGcd(m1,m2);
    if( (r1 - r2 ) % G != 0) return -1;
    LL M = m1 * m2 / G;
    // Solution exists and is unique on LCM(m1,m2) = M
    LL K = ( r2 - r1) / G;
    LL ans = ((K * m1 * coeffA ) + r1) % M;
    //Note that K * (m1*coeffA + m2*coeffB) = r2-r1
    if( ans < 0) ans += M;
    return ans;
}
```

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 $mT + nS = \gcd(m, n)$.
Prime-counting function. $\pi(n) = |\{p \leqslant n : p \text{ is prime}\}|$. $n/\ln(n) < \pi(n) < 1.3n/\ln(n)$. $\pi(1000) = 168$, $\pi(10^6) = 78498$, $\pi(10^9) = 50\,847\,534$. n-th prime $\approx n \ln n$.
Fermat primes. A Fermat prime is a prime of form $2^{2^n} + 1$. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form $2^n + 1$ is prime only if it is a Fermat prime.
Perfect numbers. $n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.
Carmichael numbers. A positive composite n is a Carmichael number ($a^{n-1} \equiv 1 \pmod n$ for all a: $\gcd(a, n) = 1$), iff n is square-free, and for all prime divisors p of n, $p - 1$ divides $n - 1$.
Number/sum of divisors. $\tau(p_1^{a_1} \ldots p_k^{a_k}) = \prod_{j=1}^{k}(a_j + 1)$. $\sigma(p_1^{a_1} \ldots p_k^{a_k}) = \prod_{j=1}^{k} \frac{p_j^{a_j+1}-1}{p_j-1}$.
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 $nu \equiv i \pmod p$.)

Pythagorean triples. Integer solutions of $x^2 + y^2 = z^2$ All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n, \gcd(m, n) = 1$ and $m \not\equiv n \pmod 2$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

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 $(a-1)(b-1) - 1 = ab - a - b$.

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.

Utilities - Number Theory.

```
LL mulmod(LL a, LL b, LL n) {
   LL res = 0;
   while (b) {
      if (b & 1) {
         --b; res = (res + a) % n;
      }
      else{
         b >>= 1;a <<= 1;
         a %= n;
      }
   }
   return res;
}
inline LL moduloSpecial(LL b, LL p, LL n) {
   LL x = 1, y = b;
   while (p) {
      if (p&1) x = mulmod(x, y, n);
      y = mulmod(y, y, n);
      p >>= 1;
   }
   return x % n;
}
bool miller(LL n, LL iteration) {
   if (n < 2) return false;
   if (n != 2 && n % 2 == 0) return false;
   LL d = n-1;
   while (d % 2 == 0) d >>= 1;
   REP(i, iteration) { // Int use 2,7,61 | LL (3.8e18) use primes till 23 |else 20 rand--Akshay
      LL a = rand() % (n - 1) + 1, temp = d;
      LL mod = moduloSpecial(a, temp, n);
      while (temp != n - 1 && mod != 1 && mod != n - 1) {
         mod = mulmod(mod, mod, n);
         temp <<= 1;
      }
      if (mod != n-1 && temp % 2 == 0) {
         return false;
      }
   }
   return true;
}
int inverse(int a, int m) {
    a = mod(a, m);
    if (a == 1) {
        return 1;
    }
    return mod((1 - m * inverse(m % a, a)) / a, m);
}
void gcdext(int &g, int &x, int &y, int a, int b) {
  if (b == 0) { g = a; x = 1; y = 0; }
  else        { gcdext(g, y, x, b, a % b); y = y - (a / b) * x; }
}

// Calculating euler Totient function.
REP(i,N) phi[i] = i;
```

```
FOR(i,2,N) if(phi[i] == i) for(int j = i ; j < N ; j += i) phi[j] -= phi[j]/i;

/* This is a pseudo random number genrator modulo n*/
LL f(LL x, LL n)
{
    LL temp = mulmod(x, x, n);
    temp = mulmod( alpha, temp, n);
    return ( temp + beta) % n;
}
/* Function returns a non trivial factor of the number N.
 * Make sure before calling this that N is not a prime using miller rabin
 */
LL pollardRho(LL N)
{
    LL x, y, d;
    while(true)
    {
        x = 2, y = 2, d = 1;
        alpha = (rand() % (N -1)) + 1;
        beta = (rand() % (N -1)) + 1;
        while( d == 1)
        {
            x = f(x,N);
            y = f( f(y,N), N);
            d = gcd( abs(x - y), N);
        }
        if( d != N) break;
    }
    assert(N % d == 0);
    // if this condition fails, consider increasing max iter and check if N is prime
    return d;
}
```

Data - Structures
String Algorithms

```
// pattern would be searched in text
#define MAXP 1000010  // length of pattern
#define MAXT 60000010 // length of text
int fault[MAXP+10];
/* fault[i] = maximum suffix of string [0-i]
which is prefix of pattern */
void setup(string &pattern,int n)
{
    int cur=0;
    fault[0]=0;
    for(int i=1;i<n;i++)
    {
        while(cur>0 && pattern[i]!=pattern[cur])
            cur=fault[cur-1];
        if(pattern[i]==pattern[cur])
            cur++;
        fault[i]=cur;
    }
}
```

```
void kmp(string &pattern,string &text)
{
    int n=pattern.size();
    int cur=0,m=text.size();
    setup(pattern,n);
    for(int i=0;i<m;i++)
    {
        while(cur>0 && text[i]!=pattern[cur])
            cur=fault[cur-1];
        if(text[i]==pattern[cur])
            cur++;
        if(cur==n)
        {
            /*** Match Found at i-n+1 (0-based) ***/
        }
    }
    //cur has the length of the suffix of text which
    //matches the prefix of pattern
}
```

```
const int N = 10010;
int A[N][26];
void setup(string p){
  for(int i=0;i<26;i++)A[0][i] = 0;
  for(int i=0;i<(int)p.size();i++){
    int k = A[i][p[i]-'a'];
    A[i][p[i]-'a'] = i + 1;
    for(int j=0;j<26;j++)A[i+1][j] = A[k][j];
  }
}
```

```
void match(string text){
  int state = 0;
  for(int i=0;i<(int)text.size();i++){
    state = A[state][text[i] - 'a'];
    cout<<state<<endl;
  }
  cout<<endl;
}
```

Suffix array.

```
int lcp[MAXN], cnt[256], cls[2][MAXN], sa[2][MAXN], st[MAXN], rnk[MAXN];
void suffixArray(string s) {
    int i, j;
    memset(cnt, 0, sizeof(cnt));
    int n = s.size();
    for (i = 0; i < n; i++) { cnt[s[i]] ++; cls[0][i] = s[i];}
    for (i = 1; i <= 255; i++) cnt[i] += cnt[i - 1];
    for (i = 0; i < n; i++) sa[0][--cnt[cls[0][i]]] = i;
    int cur = 0;
    for (i = 0; (1 << i) < n; i++) {
        int clen = (1 << i);
        for (j = 0; j < n; j++) {
            if (j > 0 && sa[cur][j - 1] + clen < n &&
                    cls[cur][sa[cur][j]] == cls[cur][sa[cur][j - 1]] &&
                 cls[cur][sa[cur][j] + clen / 2] == cls[cur][sa[cur][j - 1] + clen / 2]
            )
                cls[1 - cur][sa[cur][j]] = cls[1 - cur][sa[cur][j - 1]];
            else cls[1 - cur][sa[cur][j]] = j;
        }
        for (j = 0; j < n; j++)  { st[j] = j; sa[1 - cur][j] = sa[cur][j]; }
        for (j = 0; j < n; j++) {
            int cele = sa[cur][j] - clen;
            if (cele >= 0) sa[1 - cur][st[cls[1 - cur][cele]]++] = cele;
        }
        cur = 1 - cur;
    }
    for (i = 0; i < n; i++) { sa[0][i] = sa[cur][i]; rnk[sa[cur][i]] = i;}
    int x = 0;
    for (i = 0; i < n; i++) {
        if (rnk[i] < n - 1) {
            for (j = sa[cur][rnk[i] + 1]; max(i, j) + x < n && s[i + x] == s[j + x]; x++);
            lcp[rnk[i] + 1] = x; if (x > 0) x--;
        }
    }
}
```

Suffix Automata

```
class SAutomata {
  class SNode{public: int link, len; map<char,int> next;
    SNode(){link = -1; len = 0;}
    SNode(const SNode& other){link = other.link;len = other.len;next = other.next;}};
  vector<SNode> nodes;
  vector<long long int> dp;
  int last, cur;
  public:
  SAutomata(string S): nodes(2*S.size()),last(0),cur(1)
  {for(int i=0; i<S.size(); i++) add_char(S[i]);}
  void add_char(char A){
    int nw = cur++; nodes[nw].len = nodes[last].len+1; int p=last;
    for(; p!=-1 && nodes[p].next.find(A)==nodes[p].next.end();
    p=nodes[p].link) nodes[p].next[A]=nw; if(p==-1) nodes[nw].link = 0;
    else if(nodes[nodes[p].next[A]].len == nodes[p].len + 1)
    nodes[nw].link = nodes[p].next[A];
    else {
      int nxt = nodes[p].next[A]; int clone = cur++;
      nodes[clone] = nodes[nxt]; nodes[clone].len = nodes[p].len + 1;
      nodes[nxt].link = clone; nodes[nw].link = clone;
      for(; p!=-1 && nodes[p].next.find(A)!=nodes[p].next.end()
        && nodes[p].next[A]==nxt; p=nodes[p].link) nodes[p].next[A]=clone; }
    last = nw; }};
```

Aho-Corasick

```
class AhoCorasick{private:class Node{
friend class AhoCorasick;private:vector<int> match;
vector <unique_ptr<Node> > next;Node* blue;vector <Node*> green;public:Node():
match(),next(26),blue(nullptr),green() {}void add(string T, int match_no){if(T.empty())
  {match.PB(match_no);return;}
  if(!next[T[0]-'A']){next[T[0]-'A'] = make_unique<Node>();}
  next[T[0]-'A']->add(T.substr(1), match_no);return;}};
  unique_ptr<Node> root;int cur_nums;
  Node* cur_traversal;public:AhoCorasick():
  root(make_unique<Node>()),cur_nums(0),cur_traversal(root.get()) {}
void add(string T){root->add(T,cur_nums);cur_nums++;}void build_links()
  {root->blue = root.get();queue<Node*> qqq; //Build Blue Links
  rep(i,26){
  if(root->next[i]){root->next[i]->blue = root.get();qqq.push(root->next[i].get());}
  }while(!qqq.empty()){Node* cur = qqq.front();qqq.pop();rep(i,26)
  {Node* cur_blue = cur->blue;
while(cur->next[i] && cur_blue!=root.get() && !(cur_blue->next[i])){cur_blue=cur_blue->blue;}
  if(cur->next[i]){if(cur_blue->next[i]){cur->next[i]->blue = cur_blue->next[i].get();}
  else{cur->next[i]->blue = root.get();}qqq.push(cur->next[i].get());}}}
  //Build Green Linksrep(i,26)
  {if(root->next[i]){qqq.push(root->next[i].get());}}
  while(!qqq.empty()){Node* cur = qqq.front();qqq.pop();
  Node* cur_blue = cur->blue;if(cur_blue!=root.get()){
  if(!cur_blue->match.empty())cur->green.PB(cur_blue);
  cur->green.insert(cur->green.end(), all(cur_blue->green));}rep(i,26){if(cur->next[i]){
  qqq.push(cur->next[i].get());}}}}void reset_checker(){cur_traversal = root.get();}
  vector<int> inc_checker(char c){vector <int> ans;
  while(cur_traversal != root.get()
  && !cur_traversal->next[c-'A'])cur_traversal = cur_traversal->blue;
  if(cur_traversal->next[c-'A'])cur_traversal = cur_traversal->next[c-'A'].get();
  ans.insert(ans.end(), all(cur_traversal->match));tr(cur_traversal->green, it)
  {ans.insert(ans.end(), all((*it)->match) );}
  return ans;}};
```

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*.

Linear algebra

FFT

```
const double pi = 4*atan(1);
#define cd complex<double>
// FAST FFT
void fft(vector<cd> &a, int x = 1){
    int n = a.size();
    for (int i=1,j=n/2;i<n-1;i++){
        if (i<j) swap(a[i],a[j]);
        int p=n/2;
        while (j>=p) j-=p,p/=2;
        j+=p;}
    for (int j=2;j<=n;j*=2){
        cd wn(cos(2*pi/j*x),sin(2*pi/j*x));
        for (int i=0;i<n;i+=j){
            cd w(1,0);
            for (int k=i;k<i+j/2;k++){
                cd u=a[k],t=a[k+j/2]*w;
                a[k]=u+t;a[k+j/2]=u-t;
                w=w*wn;}}}}
void multiply(const vector<int> & a, const vector<int> & b, vector<int> & res){
    int n = 1;
    vector<cd> fa (a.begin(), a.end()), fb (b.begin(), b.end());
    while (n < 2*max (a.size(), b.size())) n <<= 1;
    fa.resize(n); fb.resize(n);
    fft (fa), fft (fb);
    for (size_t i=0; i<n; ++i)
        fa[i] = conj(fa[i] * fb[i]);
    fft (fa);
    res.resize (n);
    for (size_t i=0; i<n; ++i)
        res[i] = (int) (fa[i].real() / n + 0.5);}
```

Determinant of a Matrix

```
/*Determinant of a matrix % p.
 * P can be a prime or a power of a single prime. */
int b[200][200],n;
LL mod(LL x, LL y) {
    while(x < y) x += y;
    return x % y;}
LL deter(LL p) {
    LL coeff, r2, ret = 1;
    REP (i, n) REP (j, n) b[i][j] = mod(b[i][j], p);
    REP (i, n) {
        if (count_if(b[i] + i, b[i] + n, bind2nd(equal_to<LL>(), 0)) == n - i) {
            return 0;}
        LL g = p;
        FOR (j, i, n) g = __gcd(g, b[i][j]);
        ret = (LL)((((LL)(ret)) * g) % p);
        FOR (j, i, n) b[i][j] /= g;
        FOR (j, i, n) if (__gcd(b[i][j], p) == 1) {
            /* j can act as the pivot */
            if (j != i) {
                ret = (LL)((((LL)(ret)) * (p - 1)) % p);
                FOR (k, i, n) swap(b[k][i], b[k][j]);}
            /* now the normal elimination procedure */
            FOR (j, i + 1, n) {
                if (b[i][j]) {
                    coeff = inverse(b[i][i], p - 2, p); // Get inverse of b[i][i].
//euclide(b[i][i], p, coeff, r2);
                    coeff = (LL)((((LL)(coeff)) * b[i][j]) % p);
                    FOR(k, i, n) {
                        b[k][j] = (LL)(b[k][j] - (((LL)(b[k][i])) * coeff)%p) % p;
                        while(b[k][j] < 0) b[k][j] += p;}}}
            break;}
        ret = (LL)((((LL)(ret)) * b[i][i]) % p);}
    return ret;}
```

Rank of Boolean Matrix

```
a[] == Array of Vectors. [long long.]
int ret = n;
REP(i,n) {
    if(a[i] == 0) ret--;
    last = a[i] & -a[i];
    FOR(j,i+1,n) {
        if(a[j] & last) a[j] ^= a[i];}}
// ret == Rank of the Matrix. Rank is the size of largest set of independent vectors in a.
```

Notes

```
f[A,k] = A + A^2 + A^3 + A^4 + ... + A^k
f[A,k] = f[A,k-1] * (1 + A) - f[A,k-2] * A
int axbmodc(int a, int b,int c) { // Solves a*x = b (mod c)
        return a ? (axbmodc(c % a, (a - b % a) % a, a) * c + b) / a : 0;}
```

Inverse of a Matrix

```
typedef long double LD; #define MAXN 200
// ret = Inv(arr) Inverse of the matrix -- O(n^3) Gaussean Jordan Elimination
void findInverse(LD arr[MAXN][MAXN],LD ret[MAXN][MAXN],int N){
   LD temp[MAXN][2*MAXN];
   REP(i,N)REP(j,N)temp[i][j]=arr[i][j]; //not to disturb values in arr
   REP(i,N)REP(j,N){
          if(i!=j)temp[i][j+N]=0.0l; else temp[i][j+N]=1.0l;}
   REP(y,N){
          int maxRow=y;
          FOR(y2,y+1,N) if(fabs(temp[y2][y])>fabs(temp[maxRow][y])) maxRow=y2;
          REP(i,2*N)swap(temp[maxRow][i],temp[y][i]);
          LD base = temp[y][y];
          assert(fabs(base)>1e-9);
          FOR(y2,y+1,N){
                 LD factor = temp[y2][y]/base;
                 REP(x,2*N) temp[y2][x]-=(factor*temp[y][x]);}}
   for(int y=N-1;y>=0;y--){
          LD c=temp[y][y];
          REP(y2,y){
                 for(int x=2*N-1;x>y-1;x--){
                        temp[y2][x] -= temp[y][x] * temp[y2][y] / c;}}
          temp[y][y]/=c;
          FOR(x,N,2*N)temp[y][x]/=c;}
   REP(i,N)FOR(j,N,2*N)ret[i][j-N]=temp[i][j];
   return;}
```

Geometry

Simplex

```
//maximize c^T x subject to Ax <= b x >= 0 INPUT: A- m x n matrix;b- m-dim vector;
// c- n-dim vector; x-vector where optimal solution stored OUTPUT: value of the optimal
// solution (infinity if unbounded above, nan if infeasible)
// To use this code, create an LPSolver object with A, b, and c as arguments.Then, call Solve(x)
```

```
typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

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)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m+1][n] = 1;}
  void Pivot(int r, int s) {
    for (int i = 0; i < m+2; i++) if (i != r)
      for (int j = 0; j < n+2; j++) if (j != s)
    D[i][j] -= D[r][j] * D[i][s] / D[r][s];
    for (int j = 0; j < n+2; j++) if (j != s) D[r][j] /= D[r][s];
    for (int i = 0; i < m+2; i++) 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;
      for (int j = 0; j <= n; j++) {
    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;
      for (int i = 0; i < m; i++) {
    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);}}
  DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) 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 -numeric_limits<DOUBLE>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
    int s = -1;
    for (int j = 0; j <= n; j++)
      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 numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n+1];
    return D[m][n+1];}};
int main() {
  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {{ 6, -1, 0 },{ -1, -5, 0 },{ 1, 5, 1 },{ -1, -5, -1 }};
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };
  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);
  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);
  cerr << "VALUE: "<< value << endl;
  cerr << "SOLUTION:";
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;}
```

Pick's theorem. $I = A - B/2 + 1$, where A is the area of a lattice polygon, I is number of lattice points inside it, and B is number of lattice points on the boundary. Number of lattice points minus one on a line segment from $(0, 0)$ and (x, y) is $\gcd(x, y)$.
$a \cdot b = a_x b_x + a_y b_y = |a| \cdot |b| \cdot \cos(\theta)$
$a \times b = a_x b_y - a_y b_x = |a| \cdot |b| \cdot \sin(\theta)$
3D: $a \times b = (a_y b_z - a_z b_y,\ a_z b_x - a_x b_z,\ a_x b_y - a_y b_x)$
Line $ax + by = c$ through $A(x_1, y_1)$ and $B(x_2, y_2)$: $a = y_1 - y_2$, $b = x_2 - x_1$, $c = ax_1 + by_1$.
Half-plane to the left of the directed segment AB: $ax + by \geqslant c$.
Normal vector: (a, b). Direction vector: $(b, -a)$. Perpendicular line: $-bx + ay = d$.
Point of intersection of $a_1 x + b_1 y = c_1$ and $a_2 x + b_2 y = c_2$ is $\frac{1}{a_1 b_2 - a_2 b_1}(c_1 b_2 - c_2 b_1, a_1 c_2 - a_2 c_1)$.
Distance from line $ax + by + c = 0$ to point (x_0, y_0) is $|ax_0 + by_0 + c|/\sqrt{a^2 + b^2}$.
Distance from line AB to P (for any dimension): $\frac{|(A-P)\times(B-P)|}{|A-B|}$.
Point-line segment distance:

```
  if (dot(B-A, P-A) < 0) return dist(A,P);
  if (dot(A-B, P-B) < 0) return dist(B,P);
  return fabs(cross(P,A,B) / dist(A,B));
```

Projection of point C onto line AB is $\frac{AB \cdot AC}{AB \cdot AB} AB$.
Projection of (x_0, y_0) onto line $ax + by = c$ is $(x_0, y_0) + \frac{1}{a^2+b^2}(ad, bd)$, where $d = c - ax_0 - by_0$.
Projection of the origin is $\frac{1}{a^2+b^2}(ac, bc)$.
Segment-segment intersection. Two line segments intersect if one of them contains an endpoint of the other segment, or each segment straddles the line, containing the other segment (AB straddles line l if A and B are on the opposite sides of l.)
Circle-circle and circle-line intersection.

```
  a = x2 - x1;   b = y2 - y1;   c = [(r1^2 - x1^2 - y1^2) - (r2^2 - x2^2 - y2^2)] / 2;
  d = sqrt(a^2 + b^2);
  if not |r1 - r2| <= d <= |r1 + r2|, return "no solution"
  if d == 0, circles are concentric, a special case
  // Now intersecting circle (x1,y1,r1) with line ax+by=c
  Normalize line: a /= d; b /= d; c /= d;     // d=sqrt(a^2+b^2)
  e = c - a*x1 - b*y1;
  h = sqrt(r1^2 - e^2);                       // check if r1<e for circle-line test
  return (x1, y1) + (a*e, b*e) +/- h*(-b, a);
```

Circle from 3 points (circumcircle). Intersect two perpendicular bisectors. Line perpendicular to $ax + by = c$ has the form $-bx + ay = d$. Find d by substituting midpoint's coordinates.
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 $= \frac{2S}{a+b+c}$.
Counter-clockwise rotation around the origin. $(x, y) \mapsto (x\cos\phi - y\sin\phi, x\sin\phi + y\cos\phi)$.
90-degrees counter-clockwise rotation: $(x, y) \mapsto (-y, x)$. Clockwise: $(x, y) \mapsto (y, -x)$.
3D rotation by ccw angle ϕ around axis $\mathbf{n}$: $\mathbf{r}' = \mathbf{r}\cos\phi + \mathbf{n}(\mathbf{n} \cdot \mathbf{r})(1 - \cos\phi) + (\mathbf{n} \times \mathbf{r})\sin\phi$
Plane equation from 3 points. $N \cdot (x, y, z) = N \cdot A$, where N is normal: $N = (B - A) \times (C - A)$.

3D figures

Sphere	Volume $V = \frac{4}{3}\pi r^3$, surface area $S = 4\pi r^2$
	$x = \rho \sin\theta \cos\phi$, $y = \rho \sin\theta \sin\phi$, $z = \rho\cos\theta$, $\phi \in [-\pi, \pi]$, $\theta \in [0, \pi]$
Spherical section	Volume $V = \pi h^2(r - h/3)$, surface area $S = 2\pi rh$
Pyramid	Volume $V = \frac{1}{3} h S_{base}$
Cone	Volume $V = \frac{1}{3}\pi r^2 h$, lateral surface area $S = \pi r\sqrt{r^2 + h^2}$

Area of a simple polygon. $\frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$, where $x_n = x_0, y_n = y_0$.
Area is negative if the boundary is oriented clockwise.

Winding number. Shoot a ray from given point in an arbitrary direction. For each intersection of ray with polygon's side, add +1 if the side crosses it counterclockwise, and −1 if clockwise.

Convex Hull

```
bool operator <(PII a, PII b)
{ return a.first < b.first || (a.first == b.first && a.second < b.second); }
// IMPORTANT CROSS PRODUCT -> !!! CHANGE int TO LL if needed!!!
int cross(PII a,PII b, PII c){
    PII ab = MP(b.first-a.first,b.second-a.second);
    PII ac = MP(c.first-a.first,c.second-a.second);
    return ab.first*ac.second - ab.second*ac.first;}
// Returns convex hull in counter-clockwise order.
// Note: the last point in the returned list is the same as the first one.
vector<PII> ConvexHull(vector<PII> P) {
        int n = P.size(), k = 0; vector<PII> H(2*n);
        sort(P.begin(), P.end());
        for (int i = 0; i < n; i++)
        { while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--; H[k++] = P[i]; }
        for (int i = n-2, t = k+1; i >= 0; i--)
        { while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--; H[k++] = P[i]; }
        H.resize(k);
        return H;}
```

Computational Geometry

```
#define D_EQ(a,b)  (a>b-EPS && a<b+EPS)
#define D_LT(a,b)  ((a)<((b)-EPS))
#define D_GT(a,b)  ((a)>((b)+EPS))
typedef complex<int> Vector;
typedef complex<int> Point;
typedef pair<Point,Point> Segment;
typedef pair<Point,Point> Line;
/* Closest pair of points.
 * Complexity: O(n*log(n))
 * returns INF if number of points < 2.
 */
long double closestDistance (Point *p, int n) {
 if (n <= 1) return INF;
 sort(p, p + n, compare);
 map<long double, set<long double> > m;
 m[p[0].real()].insert(p[0].imag());
 long double h = INF, x, y, dist, hh;
 for(int i = 1; i < n; ++i) {
  hh = (long double)(sqrt(h) + 1);
  map< long double, set<long double> >::iterator sbeg = m.upper_bound(p[i].real() - hh);
  for(map<long double, set<long double> >::iterator mit = sbeg; mit != m.end(); ++mit) {
   set<long double>::iterator ybeg = (*mit).second.upper_bound( p[i].imag() - hh );
   set<long double>::iterator yend = (*mit).second.upper_bound( p[i].imag() + hh - 1);
   for (set<long double>::iterator sit = ybeg; sit!=yend; ++sit) {
    x = (*mit).first, y = (*sit);
    dist = (x - p[i].real())*(x - p[i].real()) + (y - p[i].imag())*(y - p[i].imag());
    if (dist < h) {h = dist; hh = (long double)(sqrtl(h) + 1);}}}
  m[ p[i].real() ].insert(p[i].imag());}
 return sqrtl(h);}
Point centroid(const Triangle &t) { return Point(
(t.a.real() + t.b.real() + t.c.real())/3.0, (t.a.imag() + t.b.imag() + t.c.imag())/3.0);}
pair<Point,long double> inCircle(const Triangle &t) {
 long double bc = norm(t.c - t.b), ab = norm(t.b - t.a), ac = norm(t.c - t.a);
 Point res((t.a.real() * bc + t.b.real() * ac + t.c.real() * ab),
  (t.a.imag() * bc + t.b.imag() * ac + t.c.imag() * ab));
 res /= (ab + bc + ac);
 return make_pair(res, 2.0*area(t) / (ab + bc + ac));}
Point orthoCenter(const Triangle &t) {
 Line bc;bc.first = t.b;bc.second = t.c;
 Line A_bc = perpendicular(bc, t.a);
 Line ab;ab.first = t.a;ab.second = t.b;
 Line C_ab = perpendicular(ab, t.c);
 return intersectionPointLine(A_bc, C_ab);}

/* Returns a triangle with given side lengths.
 * The vertices of the new triangle are (0,0), (a, 0) and third point in first quadrant
 */
Triangle triangleFromSideLength(long double a, long double b, long double c) {
 Point p1(0.0, 0.0), p2(a, 0.0);
 long double cos_theta = (a*a + b*b - c*c) / (2*a*b);
 long double sin_theta = sqrtl(1.0 - cos_theta*cos_theta);
 Point p3(b*cos_theta, b*sin_theta);
 Triangle res = {p1,p2,p3};
 return res;}
/**
 * CONTENTS:
 *  - struct P (point) - struct L (canonical line with integer parameters)
 *  - left turn - point inside triangle
 *  - polar angle - point inside polygon
 *  - distance from a point to a line
 *  - distance from a point to a line segment
 *  - line intersection - line segment intersection
 *  - circle through 3 points
 *  - circle of a given radius through 2 points
 *  - cut polygon (cut a convex polygon by a half-plane)
 *  - triangle area from median lengths**/
 /* Point * A simple point class used by some of the routines below. Anything else that
 supports .x and .y will work just as well. There are 2 variants - double and int.**/
struct P { double x, y; P() {}; P( double q, double w ) : x( q ), y( w ) {} };
struct P { int x, y; P() {}; P( int q, int w ) : x( q ), y( w ) {} };
 /* Line with integer parameters *
 * Represents a line through two lattice points as an
 * implicit equation:
 *  ax + by = c Stores a, b and c in lowest terms with a unique representation (positive a; if
a is 0, then positive b).Create a line by giving either (a, b, c) or a pair of points, p and q
(p == q is ok, but expect a == b == 0).Perfect for testing whether 3 or more points are collinear
- simply compute lines through all pairs of points and store them in a set or a map .**/
struct L{int a, b, c;
    void init( int A, int B, int C ){
        if( A < 0 || A == 0 && B < 0 ) { A = -A; B = -B; C = -C; }
        int d = A ?
            gcd( gcd( abs( A ), abs( B ) ), C ) :
            ( B || C ? gcd( abs( B ), C ) : 1 );
        a = A / d;
        b = B / d;
        c = C / d;}
    L() {}
    L( int A, int B, int C ) { init( A, B, C ); }
    L( P p, P q ) { init( q.y - p.y, p.x - q.x, p.x * q.y - p.y * q.x ); }
    bool operator<( const L &l ) const  //to use in a set/map{
        return a < l.a || a == l.a && ( b < l.b || b == l.b && c < l.c );}};
double dist( double ax, double ay, double bx, double by ){
    return sqrt( ( ax - bx ) * ( ax - bx ) + ( ay - by ) * ( ay - by ) );}
__typeof( P().x ) dist2( P p, P q ){
    return ( p.x - q.x ) * ( p.x - q.x ) + ( p.y - q.y ) * ( p.y - q.y );}
 /* Left turn * (this one works with integers)
 * Returns true iff the sequence v1->v2->v3 is a left turn in the plane.
   Straight line is not a left turn (change to ">= -C( EPS )").
 * #define EPS ... (1e-7 for doubles, 0 for ints)**/
template< class C >
bool leftTurn( C x1, C y1, C x2, C y2, C x3, C y3 ){
    return ( x2 - x1 ) * ( y3 - y1 ) - ( y2 - y1 ) * ( x3 - x1 ) > C( EPS );}
 /* Point inside triangle *
 * Returns true iff point (xx,yy) is inside the counter-clockwise
 * triangle (x[3],y[3])
 * REQUIRES: leftTurn()**/
bool pointInsideTriangle( double x[], double y[], double xx, double yy ){
    return leftTurn( x[0], y[0], x[1], y[1], xx, yy )
        && leftTurn( x[1], y[1], x[2], y[2], xx, yy )
        && leftTurn( x[2], y[2], x[0], y[0], xx, yy );}
 /* Polar angle
 * Returns an angle in the range [0, 2*Pi) of a given Cartesian point.
 * If the point is (0,0), -1.0 is returned.
 * P has members x and y. **/
double polarAngle( P p ){
    if( fabs( p.x ) <= EPS && fabs( p.y ) <= EPS ) return -1.0;
    if( fabs( p.x ) <= EPS ) return ( p.y > EPS ? 1.0 : 3.0 ) * acos( 0 );
    double theta = atan( 1.0 * p.y / p.x );
    if( p.x > EPS ) return( p.y >= -EPS ? theta : ( 4 * acos( 0 ) + theta ) );
    return( 2 * acos( 0 ) + theta );}
 /* Point inside polygon *
 * Returns true iff p is inside poly.
 * PRE: The vertices of poly are ordered (either clockwise or
 *      counter-clockwise.
 * POST: Modify code inside to handle the special case of "on an edge". **/
bool pointInPoly( P p, vector< P > &poly ){
    int n = poly.size();
    double ang = 0.0;
    for( int i = n - 1, j = 0; j < n; i = j++ ){
        P v( poly[i].x - p.x, poly[i].y - p.y );
        P w( poly[j].x - p.x, poly[j].y - p.y );
        double va = polarAngle( v );
        double wa = polarAngle( w );
        double xx = wa - va;
        if( va < -0.5 || wa < -0.5 || fabs( fabs( xx ) - 2 * acos( 0 ) ) < EPS ){
            // POINT IS ON THE EDGE
            assert( false );
            ang += 2 * acos( 0 );
            continue;}
        if( xx < -2 * acos( 0 ) ) ang += xx + 4 * acos( 0 );
        else if( xx > 2 * acos( 0 ) ) ang += xx - 4 * acos( 0 );
        else ang += xx;}
    return( ang * ang > 1.0 );}
 /* Distance from a point to a line. Returns the distance from p to the line defined by {a, b}.
 The closest point on the line is returned through (cpx, cpy).
 * Does not work for degenerate lines (when answer is undefined).**/
double distToLine( double ax, double ay, double bx, double by,
    double px, double py, double *cpx, double *cpy ){
    //Formula: cp = a + (p-a).(b-a) / |b-a| * (b-a)
    double proj = ( ( px - ax ) * ( bx - ax ) + ( py - ay ) * ( by - ay ) ) /
                  ( ( bx - ax ) * ( bx - ax ) + ( by - ay ) * ( by - ay ) );
    *cpx = ax + proj * ( bx - ax );
    *cpy = ay + proj * ( by - ay );
    return dist( px, py, *cpx, *cpy );}
 /* Distance from a point to a line segment. Returns the distance from p to the line segment ab.
   The closest point on ab is returned through (cpx, cpy).
   Works correctly for degenerate line segments (a == b).**/
double distToLineSegment(double ax, double ay, double bx, double by,
         double px, double py, double *cpx, double *cpy ){
    if( ( bx - ax ) * ( px - ax ) + ( by - ay ) * ( py - ay ) < EPS ){
        *cpx = ax; *cpy = ay;
        return dist( ax, ay, px, py );}
    if( ( ax - bx ) * ( px - bx ) + ( ay - by ) * ( py - by ) < EPS ){
        *cpx = bx; *cpy = by;
        return dist( bx, by, px, py );}
    return distToLine( ax, ay, bx, by, px, py, cpx, cpy );}
 /* Line intersection *
 * Returns the point of intersection of two lines:
 * (x[0],y[0])-(x[1],y[1]) and (x[2],y[2])-(x[3],y[3]). Puts the result (x, y) into (r[0], r[1])
 and returns true. If there is no intersection, return false. */
```

```
bool lineIntersect( double x[], double y[], double r[] ){
    double n[2]; n[0] = y[3] - y[2]; n[1] = x[2] - x[3];
    double denom = n[0] * ( x[1] - x[0] ) + n[1] * ( y[1] - y[0] );
    if( fabs( denom ) < EPS ) return false;
    double num = n[0] * ( x[0] - x[2] ) + n[1] * ( y[0] - y[2] );
    double t = -num / denom;
    r[0] = x[0] + t * ( x[1] - x[0] );
    r[1] = y[0] + t * ( y[1] - y[0] );
    return true;}
 /* Line segment intersection. Returns true iff two line segments:
 * (x[0],y[0])-(x[1],y[1]) and (x[2],y[2])-(x[3],y[3])
 * intersect. Call lineIntersect( x, y ) to get the point of intersection.
 * WARNING: Does not work for collinear line segments!**/
template< class T >
bool lineSegIntersect( vector< T > &x, vector< T > &y ){
    double ucrossv1 = ( x[1] - x[0] ) * ( y[2] - y[0] ) - ( y[1] - y[0] ) * ( x[2] - x[0] );
    double ucrossv2 = ( x[1] - x[0] ) * ( y[3] - y[0] ) - ( y[1] - y[0] ) * ( x[3] - x[0] );
    if( ucrossv1 * ucrossv2 > 0 ) return false;
    double vcrossu1 = ( x[3] - x[2] ) * ( y[0] - y[2] ) - ( y[3] - y[2] ) * ( x[0] - x[2] );
    double vcrossu2 = ( x[3] - x[2] ) * ( y[1] - y[2] ) - ( y[3] - y[2] ) * ( x[1] - x[2] );
    return( vcrossu1 * vcrossu2 <= 0 );}
 /* Circle through 3 points * Computes the circle containing the 3 given points.
 The 3 points are (x[0], y[0]), (x[1], y[1]) and (x[2], y[2]). The centre of the circle
 is returned as (r[0], r[1]). The radius is returned normally. If the circle is
undefined (the points are collinear), -1.0 is returned.**/
double circle3pts( double x[], double y[], double r[] ){
    double lix[4], liy[4];
    lix[0] = 0.5 * ( x[0] + x[1] ); liy[0] = 0.5 * ( y[0] + y[1] );
    lix[1] = lix[0] + y[1] - y[0];  liy[1] = liy[0] + x[0] - x[1];
    lix[2] = 0.5 * ( x[1] + x[2] ); liy[2] = 0.5 * ( y[1] + y[2] );
    lix[3] = lix[2] + y[2] - y[1];  liy[3] = liy[2] + x[1] - x[2];
    if( !lineIntersect( lix, liy, r ) ) return -1.0;
    return sqrt(
        ( r[0] - x[0] ) * ( r[0] - x[0] ) +
        ( r[1] - y[0] ) * ( r[1] - y[0] ) );}
 /* Circle of a given radius through 2 points. Computes the center of a circle
    containing the 2 given points. The circle has the given radius. The returned
    center is never to the right of the vector (x1, y1)-->(x2, y2). If this is
    possible, returns true and passes the center through the ctr array. Otherwise, returns false.**/
bool circle2ptsRad( double x1, double y1, double x2, double y2, double r, double ctr[2] ){
    double d2 = ( x1 - x2 ) * ( x1 - x2 ) + ( y1 - y2 ) * ( y1 - y2 );
    double det = r * r / d2 - 0.25;
    if( det < 0.0 ) return false;
    double h = sqrt( det );
    ctr[0] = ( x1 + x2 ) * 0.5 + ( y1 - y2 ) * h;
    ctr[1] = ( y1 + y2 ) * 0.5 + ( x2 - x1 ) * h;
    return true;}
 /* Cut Polygon *
 * Intersects a given convex polygon with a half-plane.
   The half-plane is defined as the one on the left side of the directed line a-->b.
   The polygon 'poly' is modified. The half-plane is considered open, so
   if only one vertex of the polygon remains after the cut, it is eliminated.**/
template< class Pt >
void cutPoly( list< Pt > &poly, Pt a, Pt b ){
    if( !poly.size() ) return;
    // see if the last point of the polygon is inside
    bool lastin = leftTurn( a, b, poly.back() );
    // find the boundary points
    __typeof( poly.begin() ) fi = poly.end(), la = fi, fip = fi, lan = fi;
    for( __typeof( fi ) i = --poly.end(), j = poly.begin();
        j != poly.end(); i = j++ ){
        int thisin = leftTurn( a, b, *j );
        if( lastin && !thisin ) { la = i; lan = j; }
        if( !lastin && thisin ) { fi = j; fip = i; }
        lastin = thisin;}
    // see if we have crossed the line at all
    if( fi == poly.end() ){
        if( !lastin ) poly.clear();
        return;}
    // if we cut off a corner, insert a new point
    if( lan == fip ){
        poly.insert( lan, *lan );
        --lan;}
    // compute intersection points
    Pt r;
    lineIntersect( *la, *lan, a, b, r );
    *lan = r;
    lineIntersect( *fip, *fi, a, b, r );
    *fip = r;
    // erase the part that disappears
    __typeof( fi ) i = lan; ++i;
    while( i != fip ){
        if( i == poly.end() ) { i = poly.begin(); if( i == fip ) break; }
        poly.erase( i++ );}
    // clean up duplicate points
    if( dist2( *lan, *fip ) < EPS ) poly.erase( fip );}
 /* Triangle Area from Medians * Given the lengths of the 3 medians of a
 triangle, returns the triangle's area, or -1 if it impossible.
 * WARNING: Deal with the case of zero area carefully. **/
double triAreaFromMedians( double ma, double mb, double mc ){
    double x = 0.5 * ( ma + mb + mc );
    double a = x * ( x - ma ) * ( x - mb ) * ( x - mc );
    if( a < 0.0 ) return -1.0;
    else return sqrt( a ) * 4.0 / 3.0;}
 /* Great Circle * Given two pairs of (latitude, longitude), returns the
 * great circle distance between them.**/
double greatCircle( double laa, double loa, double lab, double lob ){
    double PI = acos( -1.0 ), R = 6378.0;
    double u[3] = { cos( laa ) * sin( loa ), cos( laa ) * cos( loa ), sin( laa ) };
    double v[3] = { cos( lab ) * sin( lob ), cos( lab ) * cos( lob ), sin( lab ) };
    double dot = u[0]*v[0] + u[1]*v[1] + u[2]*v[2]; bool flip = false;
    if( dot < 0.0 ){flip = true;for( int i = 0; i < 3; i++ ) v[i] = -v[i];}
    double cr[3] = { u[1]*v[2] - u[2]*v[1], u[2]*v[0] - u[0]*v[2], u[0]*v[1] - u[1]*v[0] };
    double theta = asin( sqrt( cr[0]*cr[0] + cr[1]*cr[1] + cr[2]*cr[2] ) );
    double len = theta * R;
    if( flip ) len = PI * R - len;
    return len;}
```

/* Tangent to 2 circles */
$C_1 = (x_1, y_1, r_1), C_2 = (x_2, y_2, r_2)$
$\Delta x = x_2 - x_1, \Delta y = y_2 - y_1, \Delta r = r_2 - r_1$
$X = \Delta x/d, Y = \Delta y/d, R = \Delta r/d$
$tangent => ax + by + c = 0,$
$a = RX - kY\sqrt{(1 - R^2)}$
$b = RY + kX\sqrt{(1 - R^2)}$
$c = r_1 - (ax_1 + by_1)$
$k = \pm 1$

```
// Finding inverse of i mod m.
inv[1] = 1;
for (int i=2; i<m; ++i) inv[i] = (m - (m/i) * inv[m%i] % m) % m;

// Grey Code.
int g (int n) { return n ^ (n >> 1);}
int rev_g (int g) {int n = 0;for (; g; g>>=1)n ^= g; return n;}

// Find primes and also lp[i] is smallest prime divisor of i in O(N).
const int N = 10000000;
int lp[N+1]; VI pr;
for(int i=2;i<=N;++i){
  if(lp[i]==0){lp[i] = i;pr.PB(i);}
  for(int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j]<=N; ++j)
    lp[i * pr[j]] = pr[j];}

const int INF = 1000000000;
class segtree {
    public:
        vector<int> ST,LZ;
        int base,ql,qr;
        segtree(int N){
            base = 1; while(base<N)base*=2;
            ST.resize(2*base);LZ.resize(2*base);}
        void up(int id){
            ST[id] = min(ST[2*id] , ST[2*id+1]);}
        void push(int id){
            ST[id]+=LZ[id];
            if(id<base)
                LZ[2*id] += LZ[id],
                    LZ[1 + 2*id] += LZ[id];
            LZ[id] = 0;}
        void update(int val, int id, int l, int r){
            push(id); if(ql>=r or qr<=l)return;
            if(l>=ql and r<=qr){LZ[id]+=val; push(id); return;}
            update(val,2*id,l,(l+r)/2);
            update(val,1 + 2*id,(l+r)/2,r);
            up(id);}
        int query(int id, int l, int r){
            push(id);
            if(l>=qr or r<=ql)return INF;
            if(l>=ql and r<=qr)return ST[id];
            return min(query(2*id,l,(l+r)/2) , query(2*id+1,(l+r)/2,r));}
        void update(int l, int r, int val){
            ql = l; qr = r+1; update(val,1,0,base);}
        int query(int l, int r){
            ql = l; qr = r+1; return query(1,0,base);}};

//Treap BST
typedef struct node{
       int val,prior,size;
       struct node *l,*r;}node;
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;//elem=key comes 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 temp=t;merge(t,t->l,t->r);free(temp);}
       else erase(t->val<key?t->r:t->l,key);upd_sz(t);}
void unite (pnode &t,pnode l, pnode r) {
if(!l||!r)returnvoid(t=l?l:r); if (l->prior < r->prior) swap (l, r); pnode lt, rt;
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);

//Treap Interval
typedef struct node{int prior,size; int val;//value stored in the array
int sum;//whatever info you want to maintain in segtree for each node
int lazy;//whatever lazy update you want to do
struct node *l,*r;}node;
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 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;//no need to reset lazy coz when we call this
           //lazy would itself be propagated}
void combine(pnode& t,pnode l,pnode r){//combining two ranges of segtree
       if(!l || !r)return void(t = l?l:r);t->sum = l->sum + r->sum;}
void operation(pnode t){//operation of segtree
       if(!t)return;
       reset(t);//reset the value of current node assuming it now
           //represents a single element of the array
       lazy(t->l);lazy(t->r);//imp:propagate lazy before combining t->l,t->r;
       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 curr_pos = add + sz(t->l);
       if(curr_pos<=pos)//element at pos goes to left subtree(l)
split(t->r,t->r,r,pos,curr_pos+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){//l->leftarray,r->rightarray,t->resulting 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);}
```