Team Notebook

Contents

1 Dynamic Programming

1.1 Convex Hull Trick

```
// Complexity: O(nlog(n))

template <class T>
    struct ConvexHull {
        int head, tail;
        T A[maxn], B[maxn];

        ConvexHull(): head(0), tail(0) {}

        bool bad(int l1, int l2, int l3) {
            return 1.0 * (B[l3] - B[l1]) / (A[l1] - A[l3]) < 1.0 * (B[l2] - B[l1]) / (
                A[l1] - A[l2]);
        }

        void add(T a, T b) {
            A[tail] = a; B[tail++] = b;
            while (tail > 2 && bad(tail - 3, tail - 2, tail - 1)) {
                A[tail - 2] = A[tail - 1];
                B[tail - 2] = B[tail - 1];
                tail--;
            }
        }

        T get(T x) {
            int l = 0, r = tail - 1;
            while (l < r) {
                int m = (l + r) / 2;
                long long f1 = A[m] * x + B[m];
                long long f2 = A[m + 1] * x + B[m + 1];
                if (f1 <= f2) l = m + 1;
                else r = m;
            }
            return A[l] * x + B[l];
        }
    };
```

2 Data Structure

2.1 BIT2D

```
vector<int> nodes[maxn];
vector<int> f[maxn];

void fakeUpdate(int u, int v) {
    for(int x = u; x <= n; x += x & -x)
        nodes[x].push_back(v);
}

void fakeGet(int u, int v) {
    for(int x = u; x > 0; x -= x & -x)
        nodes[x].push_back(v);
}

void update(int u, int v) {
    for(int x = u; x <= n; x += x & -x)
        for(int y = lower_bound(nodes[x].begin(), nodes[x].end(), v) - nodes[x].begin
            () + 1; y <= nodes[x].size(); y += y & -y)
            f[x][y]++;
}

int get(int u, int v) {
    int res = 0;
    for(int x = u; x > 0; x -= x & -x)
        for(int y = upper_bound(nodes[x].begin(), nodes[x].end(), v) - nodes[x].begin
            (); y > 0; y -= y & -y)
            res += f[x][y];
    return res;
}
```

2.2 kdTree

```
// -----------------------------------------------------------------
// A straightforward, but probably sub-optimal KD-tree implmentation
// that's probably good enough for most things (current it's a
// 2D-tree)
//
//  - constructs from n points in O(n lg^2 n) time
//  - handles nearest-neighbor query in O(lg n) if points are well
//    distributed
//  - worst case for nearest-neighbor may be linear in pathological
//    case
//
// Sonny Chan, Stanford University, April 2009
// -----------------------------------------------------------------

#include <bits/stdc++.h>
using namespace std;

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

bool operator==(const point &a, const point &b) {
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b) {
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b) {
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b) {
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox {
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < (int) v.size(); ++i) {
            x0 = min(x0, v[i].x);   x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y);   y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)       return pdist2(point(x0, y0), p);
            else if (p.y > y1)  return pdist2(point(x0, y1), p);
            else                return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)       return pdist2(point(x1, y0), p);
            else if (p.y > y1)  return pdist2(point(x1, y1), p);
            else                return pdist2(point(x1, p.y), p);
        }
        else {
            if (p.y < y0)       return pdist2(point(p.x, y0), p);
            else if (p.y > y1)  return pdist2(point(p.x, y1), p);
            else                return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or leaf
struct kdnode {
    bool leaf;      // true if this is a leaf node (has one point)
    point pt;       // the single point of this is a leaf
    bbox bound;     // bounding box for set of points in children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete second; }

    // intersect a point with this node (returns squared distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp) {
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        } else {
            // split on x if the bbox is wider than high (not best heuristic...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode();   first->construct(vl);
            second = new kdnode();  second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree {
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to nearest point
    ntype search(kdnode *node, const point &p) {
        if (node->leaf) {
            // commented special case tells a point not to find itself
//            if (p == node->pt) return sentry;
//            else
                return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
```

```
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        } else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};

// --------------------------------------------------------------------------
// some basic test code here

int main() {
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i) {
        vp.push_back(point(rand()%100000, rand()%100000));
    }
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ", " << q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }

    return 0;
}

// --------------------------------------------------------------------------
```

2.3 HeavyLight

```
// Usage:
// dfs_sz(1)
// dfs_hld(1)
//
// T(v) = [in[v]; out[v])
// path from v -> last vertexn ascending heavy path from v (next[v]): [in[next[v]]; in
    [v]]
void dfs_sz(int v) {
    sz[v] = 1;
    for(auto &u: g[v]) {
        dfs_sz(u);
        sz[v] += sz[u];
        if(sz[u] > sz[g[v][0]])
            swap(u, g[v][0]);
    }
}

void dfs_hld(int v) {
    in[v] = t++;
    rin[in[v]] = v;
    for(auto u: g[v]) {
        nxt[u] = (u == g[v][0] ? nxt[v] : u);
        dfs_hld(u);
    }
    out[v] = t;
```

3 Dynamic Programming

3.1 Convex Hull Trick

```
// Complexity: O(nlog(n))

template <class T>
    struct ConvexHull {
        int head, tail;
        T A[maxn], B[maxn];

        ConvexHull(): head(0), tail(0) {}

        bool bad(int l1, int l2, int l3) {
            return 1.0 * (B[l3] - B[l1]) / (A[l1] - A[l3]) < 1.0 * (B[l2] - B[l1]) / (
                 A[l1] - A[l2]);
        }

        void add(T a, T b) {
            A[tail] = a; B[tail++] = b;
            while (tail > 2 && bad(tail - 3, tail - 2, tail - 1)) {
                A[tail - 2] = A[tail - 1];
                B[tail - 2] = B[tail - 1];
                tail--;
            }
        }

        T get(T x) {
            int l = 0, r = tail - 1;
            while (l < r) {
                int m = (l + r) / 2;
                long long f1 = A[m] * x + B[m];
                long long f2 = A[m + 1] * x + B[m + 1];
                if (f1 <= f2) l = m + 1;
                else r = m;
            }
            return A[l] * x + B[l];
        }
    };
```

4 Graph

4.1 2

```
// Tested:
// - http://codeforces.com/contest/568/problem/C
// - https://open.kattis.com/contests/nwerc15open/problems/cleaningpipes
const int MN = 200111;   // 2 * no variables.
int n;
vector<int> g[MN], gt[MN];
vector<bool> used;
vector<int> order, comp;

void dfs1 (int v) {
    used[v] = true;
    for (size_t i=0; i<g[v].size(); ++i) {
        int to = g[v][i];
        if (!used[to])
            dfs1 (to);
        }
        order.push_back (v);
}
void dfs2 (int v, int cl) {
    comp[v] = cl;
    for (size_t i=0; i<gt[v].size(); ++i) {
        int to = gt[v][i];
        if (comp[to] == -1)
            dfs2 (to, cl);
        }
```

```
}
int main() {
    // n = 2 * (number of boolean variables)
    // NOTE: if we need to fix some variable, e.g. set i = 0 --> addEdge(2*i+1, 2*i)
    // var i --> 2 nodes: 2*i, 2*i+1.

    n = 200000;  // 2 * number of variables.
    used.clear();
    order.clear();
    comp.clear();

    REP(i,n) {
        g[i].clear();
        gt[i].clear();
    }

    // for each condition:
    // u -> v: addEdge(u, v)

    used.assign (n, false);
    REP(i,n) if (!used[i]) dfs1 (i);
    comp.assign (n, -1);
    for (int i=0, j=0; i<n; ++i) {
        int v = order[n-i-1];
        if (comp[v] == -1) dfs2 (v, j++);
    }
    REP(i,n) if (comp[i] == comp[i^1]) {
        puts ("NO SOLUTION"); return 0;
    }
    for (int i=0; i<n; i += 2) {
        int ans = comp[i] > comp[i^1] ? i : i^1;
        printf ("%d ", ans);
    }
}
```

4.2 Biconnected Components

```
const int N = 1024;

int count, parent[N], n; //n vertices 0..n-1
bool visited[N];
vector<int> G[N];
stack<pair<int, int> > s;

void OutputComp(int u, int v) {
    pair<int, int> edge;
    do {
        edge = s.top(); s.pop();
        printf("%d %d\n", edge.first, edge.second);
    } while (edge != make_pair(u, v));
    printf("\n");
}

void dfs(int u) {
    visited[u] = true;
    count++;
    low[u] = num[u] = count;
    for (int v : G[u]) {
        if (!visited[v]) {
            s.push({u, v});
            parent[v] = u;
            dfs(v);
            if (low[v] > num[u]) OutputComp(u, v);
            low[u] = min(low[u], low[v]);
        } else if (parent[u] != v && num[v] < num[u]) {
            s.push({u, v});
            low[u] = min(low[u], num[v]);
        }
    }
}

void BiconnectedComponents {
    count = 0;
    memset(parent, -1, sizeof parent);
    for (int i = 0; i < n; i++)
        if (!visited[i]) dfs(i);
}
```

4.3 Euler Path

```
// NOTES:
// - When choosing starting vertex (for calling find_path), make sure deg[start] > 0.
// - If find Euler path, starting vertex must have odd degree.
// - Check no solution: SZ(path) == nEdge + 1.
//
// Tested:
// - https://open.kattis.com/problems/eulerianpath (directed)
// - SGU 101 (undirected).
//
// If directed:
// - Edge --> int
// - add_edge(int a, int b) { adj[a].push_back(b); }
// - Check for no solution:
// - - for all u, |in_deg[u] - out_deg[u]| <= 1
// - - At most 1 vertex with in_deg[u] - out_deg[u] = 1
// - - At most 1 vertex with out_deg[u] - in_deg[u] = 1 (start vertex)
// - - BFS from start vertex, all vertices u with out_deg[u] > 0 must be visited
struct Edge {
    int to;
    list<Edge>::iterator rev;

    Edge(int to) :to(to) {}
};

const int MN = 100111;
list<Edge> adj[MN];
vector<int> path; // our result

void find_path(int v) {
    while(adj[v].size() > 0) {
        int vn = adj[v].front().to;
        adj[vn].erase(adj[v].front().rev);
        adj[v].pop_front();
        find_path(vn);
    }
    path.push_back(v);
}

void add_edge(int a, int b) {
    adj[a].push_front(Edge(b));
    auto ita = adj[a].begin();
    adj[b].push_front(Edge(a));
    auto itb = adj[b].begin();
    ita->rev = itb;
    itb->rev = ita;
}
```

4.4 Maximum Bipartite Matching

```
// Maximum bipartite matching
// Index from 1
// Find max independent set:
// for(i = 1 -> M) if (mat.matchL[i] > 0) {
//   if (mat.dist[i] < inf) {
//     for(j = 1 -> N) if (ke[i][j]) right.erase(j); }
//   else left.erase(i);
// }
// Find vertices that belong to all maximum matching:
// - L = vertices not matched on left side --> BFS from these vertices
//   (left --> right: unmatched edges, right --> left: matched edges)
//   reachable vertices on left side --> not belong to some maximum matching
// - Do similar for right side
// Tested:
// - http://codeforces.com/gym/100216 - J
// - SRM 589 - 450
// - http://codeforces.com/gym/100337 - A
const int inf = 1000111;
```

```
struct Matching {
    int n;
    vector<int> matchL, matchR, dist;
    vector<bool> seen;
    vector< vector<int> > ke;

    Matching(int n) : n(n), matchL(n+1), matchR(n+1), dist(n+1), seen(n+1, false), ke(
         n+1) {}

    void addEdge(int u, int v) {
        ke[u].push_back(v);
    }

    bool bfs() {
        queue<int> qu;
        for(int u = 1; u <= n; ++u)
            if (!matchL[u]) {
                dist[u] = 0;
                qu.push(u);
            } else dist[u] = inf;
        dist[0] = inf;

        while (!qu.empty()) {
            int u = qu.front(); qu.pop();
            for(__typeof(ke[u].begin()) v = ke[u].begin(); v != ke[u].end(); ++v) {
                if (dist[matchR[*v]] == inf) {
                    dist[matchR[*v]] = dist[u] + 1;
                    qu.push(matchR[*v]);
                }
            }
        }
        return dist[0] != inf;
    }

    bool dfs(int u) {
        if (u) {
            for(__typeof(ke[u].begin()) v = ke[u].begin(); v != ke[u].end(); ++v)
                if (dist[matchR[*v]] == dist[u] + 1 && dfs(matchR[*v])) {
                    matchL[u] = *v;
                    matchR[*v] = u;
                    return true;
                }
            dist[u] = inf;
            return false;
        }
        return true;
    }

    int match() {
        int res = 0;
        while (bfs()) {
            for(int u = 1; u <= n; ++u)
                if (!matchL[u])
                    if (dfs(u)) ++res;
        }
        return res;
    }
};
```

```
/// Max independent set tracing

#include <stdio.h>
#include <string.h>
#include <queue>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

#define long long long
#define f1(i,n) for (int i=1; i<=n; i++)
#define f0(i,n) for (int i=0; i<n; i++)

#define N 2003
int m, n, q;
vector<int> a[N]; //
int Assigned[N], Visited[N]; //
bool Choosed[N]; //

bool visit(int u, int Key){
   if (Visited[u]==Key) return false; Visited[u]=Key;
   for (int i=0; int v=a[u][i]; i++)
   if (!Assigned[v] || visit(Assigned[v], Key))
   { Assigned[u]=v; Assigned[v]=u; return true; }
   return false;
}

void konig(){
   queue<int> qu;

   f1(i,m) if (!Assigned[i]) qu.push(i);
   f1(i,n) if (!Assigned[N-i]) qu.push(N-i);

   while (qu.size()){
      int u=qu.front(); qu.pop();
      for (int i=0; int v=a[u][i]; i++)
      if (!(Choosed[v]++)) qu.push(Assigned[v]);
   }

   f1(i,m) if (Assigned[i] && !Choosed[i] && !Choosed[Assigned[i]])
   Choosed[i]=true;
}

main(){
   scanf("%d%d%d", &m, &n, &q);
   if (m+n+q==0) return 0;
   f1(i,q){
      int x, y;
      scanf("%d%d", &x, &y);
      a[x].push_back(N-y);
      a[N-y].push_back(x);
   }
   f1(i,m) a[i].push_back(0);
   f1(i,n) a[N-i].push_back(0);

   static int cnt=0; int Count=0;
   f1(i,m) if (!Assigned[i]) visit(i, ++cnt);
   f1(i,m) if (Assigned[i]) Count++;
   cout << Count;

   konig();
   f1(i,m) if (Choosed[i]) printf(" r%d", i);
   f1(i,n) if (Choosed[N-i]) printf(" c%d", i);
   printf("\n");

   f1(i,m) a[i].clear();
   f1(i,n) a[N-i].clear();
   memset(Assigned, 0, sizeof Assigned);
   memset(Choosed, 0, sizeof Choosed);

   main();
}
```

4.5 General Matching

```
// General matching on graph
// Notes:
// - Index from 1
// - Must add edges in both directions.

const int maxv = 1000;
const int maxe = 50000;

struct EdmondsLawler {
    int n, E, start, finish, newRoot, qsize, adj[maxe], next[maxe], last[maxv], mat[
         maxv], que[maxv], dad[maxv], root[maxv];
    bool inque[maxv], inpath[maxv], inblossom[maxv];

    void init(int _n) {
        n = _n; E = 0;
        for(int x=1; x<=n; ++x) { last[x] = -1; mat[x] = 0; }
```

```
}
void add(int u, int v) {
    adj[E] = v; next[E] = last[u]; last[u] = E++;
}
int lca(int u, int v) {
    for(int x=1; x<=n; ++x) inpath[x] = false;
    while (true) {
        u = root[u];
        inpath[u] = true;
        if (u == start) break;
        u = dad[mat[u]];
    }
    while (true) {
        v = root[v];
        if (inpath[v]) break;
        v = dad[mat[v]];
    }
    return v;
}
void trace(int u) {
    while (root[u] != newRoot) {
        int v = mat[u];

        inblossom[root[u]] = true;
        inblossom[root[v]] = true;

        u = dad[v];
        if (root[u] != newRoot) dad[u] = v;
    }
}
void blossom(int u, int v) {
    for(int x=1; x<=n; ++x) inblossom[x] = false;

    newRoot = lca(u, v);
    trace(u); trace(v);

    if (root[u] != newRoot) dad[u] = v;
    if (root[v] != newRoot) dad[v] = u;

    for(int x=1; x<=n; ++x) if (inblossom[root[x]]) {
        root[x] = newRoot;
        if (!inque[x]) {
            inque[x] = true;
            que[qsize++] = x;
        }
    }
}
bool bfs() {
    for(int x=1; x<=n; ++x){
        inque[x] = false;
        dad[x] = 0;
        root[x] = x;
    }
    qsize = 0;
    que[qsize++] = start;
    inque[start] = true;
    finish = 0;

    for(int i=0; i<qsize; ++i) {
        int u = que[i];
        for (int e = last[u]; e != -1; e = next[e]) {
            int v = adj[e];
            if (root[v] != root[u] && v != mat[u]) {
                if (v == start || (mat[v] > 0 && dad[mat[v]] > 0)) blossom(u, v);
                else if (dad[v] == 0) {
                    dad[v] = u;
                    if (mat[v] > 0) que[qsize++] = mat[v];
                    else {
                        finish = v;
                        return true;
                    }
                }
            }
        }
    }
    return false;
}
void enlarge() {
    int u = finish;
        while (u > 0) {
            int v = dad[u], x = mat[v];
            mat[v] = u;
            mat[u] = v;
            u = x;
        }
    }
    int maxmat() {
        for(int x=1; x<=n; ++x) if (mat[x] == 0) {
            start = x;
            if (bfs()) enlarge();
        }

        int ret = 0;
        for(int x=1; x<=n; ++x) if (mat[x] > x) ++ret;
        return ret;
    }
} edmonds;
```

4.6 Max Flow

```
// Source: e-maxx.ru
// Tested with: VOJ - NKFLOW, VOJ - MCQUERY (Gomory Hu)

// Usage:
// MaxFlow flow(n)
// For each edge: flow.addEdge(u, v, c)
// Index from 0

// Tested:
// - https://open.kattis.com/problems/maxflow
const int INF = 1000000000;

struct Edge {
    int a, b, cap, flow;
};

struct MaxFlow {
    int n, s, t;
    vector<int> d, ptr, q;
    vector< Edge > e;
    vector< vector<int> > g;

    MaxFlow(int n) : n(n), d(n), ptr(n), q(n), g(n) {
        e.clear();
        REP(i,n) {
            g[i].clear();
            ptr[i] = 0;
        }
    }

    void addEdge(int a, int b, int cap) {
        Edge e1 = { a, b, cap, 0 };
        Edge e2 = { b, a, 0, 0 };
        g[a].push_back( (int) e.size() );
        e.push_back(e1);
        g[b].push_back( (int) e.size() );
        e.push_back(e2);
    }
    int getMaxFlow(int _s, int _t) {
        s = _s; t = _t;
        int flow = 0;
        for (;;) {
            if (!bfs()) break;
            REP(i,n) ptr[i] = 0;
            while (int pushed = dfs(s, INF))
                flow += pushed;
        }
        return flow;
    }

private:
    bool bfs() {
        int qh = 0, qt = 0;
        q[qt++] = s;
        REP(i,n) d[i] = -1;
```

```
        d[s] = 0;

        while (qh < qt && d[t] == -1) {
            int v = q[qh++];
            REP(i,g[v].size()) {
                int id = g[v][i], to = e[id].b;
                if (d[to] == -1 && e[id].flow < e[id].cap) {
                    q[qt++] = to;
                    d[to] = d[v] + 1;
                }
            }
        }
        return d[t] != -1;
    }

    int dfs (int v, int flow) {
        if (!flow) return 0;
        if (v == t) return flow;
        for (; ptr[v] < (int)g[v].size(); ++ptr[v]) {
            int id = g[v][ptr[v]],
                to = e[id].b;
            if (d[to] != d[v] + 1) continue;
            int pushed = dfs(to, min(flow, e[id].cap - e[id].flow));
            if (pushed) {
                e[id].flow += pushed;
                e[id^1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
};
```

4.7 Min Cost

```
// Min Cost Max Flow - SPFA
// Index from 0
// edges cap changed during find flow
// Lots of double comparison --> likely to fail for double
// Example:
// MinCostFlow mcf(n);
// mcf.addEdge(u, v, cap, cost);
// cout << mcf.minCostFlow() << endl;
// Tested:
// - https://open.kattis.com/problems/mincostmaxflow
// - http://codeforces.com/gym/100213 - A
// - http://codeforces.com/gym/100216 - A
// - http://codeforces.com/gym/100222 - D
// - ACM Regional Daejeon 2014 - L (negative weights)
// - http://www.infoarena.ro/problema/fmcm (TLE 3 tests)
// - https://codeforces.com/contest/277/problem/E

template<class Flow=int, class Cost=int>
struct MinCostFlow {
    const Flow INF_FLOW = 1000111000;
    const Cost INF_COST = 1000111000111000LL;

    int n, t, S, T;
    Flow totalFlow;
    Cost totalCost;
    vector<int> last, visited;
    vector<Cost> dis;
    struct Edge {
        int to;
        Flow cap;
        Cost cost;
        int next;
        Edge(int to, Flow cap, Cost cost, int next) :
                to(to), cap(cap), cost(cost), next(next) {}
    };
    vector<Edge> edges;

    MinCostFlow(int n) : n(n), t(0), totalFlow(0), totalCost(0), last(n, -1), visited(
         n, 0), dis(n, 0) {
        edges.clear();
    }

    int addEdge(int from, int to, Flow cap, Cost cost) {
        edges.push_back(Edge(to, cap, cost, last[from]));
        last[from] = t++;
        edges.push_back(Edge(from, 0, -cost, last[to]));
        last[to] = t++;
        return t - 2;
    }

    pair<Flow, Cost> minCostFlow(int _S, int _T) {
        S = _S; T = _T;
        SPFA();
        while (1) {
            while (1) {
                REP(i,n) visited[i] = 0;
                if (!findFlow(S, INF_FLOW)) break;
            }
            if (!modifyLabel()) break;
        }
        return make_pair(totalFlow, totalCost);
    }

private:
    void SPFA() {
        REP(i,n) dis[i] = INF_COST;
        priority_queue< pair<Cost,int> > Q;
        Q.push(make_pair(dis[S]=0, S));
        while (!Q.empty()) {
            int x = Q.top().second;
            Cost d = -Q.top().first;
            Q.pop();
            // For double: dis[x] > d + EPS
            if (dis[x] != d) continue;
            for(int it = last[x]; it >= 0; it = edges[it].next)
                if (edges[it].cap > 0 && dis[edges[it].to] > d + edges[it].cost)
                    Q.push(make_pair(-(dis[edges[it].to] = d + edges[it].cost), edges[
                         it].to));
        }
        Cost disT = dis[T]; REP(i,n) dis[i] = disT - dis[i];
    }

    Flow findFlow(int x, Flow flow) {
        if (x == T) {
            totalCost += dis[S] * flow;
            totalFlow += flow;
            return flow;
        }
        visited[x] = 1;
        Flow now = flow;
        for(int it = last[x]; it >= 0; it = edges[it].next)
            // For double: fabs(dis[edges[it].to] + edges[it].cost - dis[x]) < EPS
            if (edges[it].cap && !visited[edges[it].to] && dis[edges[it].to] + edges[
                 it].cost == dis[x]) {
                Flow tmp = findFlow(edges[it].to, min(now, edges[it].cap));
                edges[it].cap -= tmp;
                edges[it ^ 1].cap += tmp;
                now -= tmp;
                if (!now) break;
            }
        return flow - now;
    }

    bool modifyLabel() {
        Cost d = INF_COST;
        REP(i,n) if (visited[i])
            for(int it = last[i]; it >= 0; it = edges[it].next)
                if (edges[it].cap && !visited[edges[it].to])
                    d = min(d, dis[edges[it].to] + edges[it].cost - dis[i]);

        // For double: if (d > INF_COST / 10)     INF_COST = 1e20
        if (d == INF_COST) return false;
        REP(i,n) if (visited[i])
            dis[i] += d;
        return true;
    }
};
```

4.8 Global Min Cut

```
// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//
// OUTPUT:
//     - (min cut value, nodes in half of min cut)

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
  int N = weights.size();
  VI used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    VI w = weights[0];
    VI added = used;
    int prev, last = 0;
    for (int i = 0; i < phase; i++) {
      prev = last;
      last = -1;
      for (int j = 1; j < N; j++)
        if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
      if (i == phase-1) {
        for (int j = 0; j < N; j++) weights[prev][j] += weights[last][j];
        for (int j = 0; j < N; j++) weights[j][prev] = weights[prev][j];
        used[last] = true;
        cut.push_back(last);
        if (best_weight == -1 || w[last] < best_weight) {
          best_cut = cut;
          best_weight = w[last];
        }
      } else {
        for (int j = 0; j < N; j++)
          w[j] += weights[last][j];
        added[last] = true;
      }
    }
  }
  return make_pair(best_weight, best_cut);
}

// BEGIN CUT
// The following code solves UVA problem #10989: Bomb, Divide and Conquer
int main() {
  int N;
  cin >> N;
  for (int i = 0; i < N; i++) {
    int n, m;
    cin >> n >> m;
    VVI weights(n, VI(n));
    for (int j = 0; j < m; j++) {
      int a, b, c;
      cin >> a >> b >> c;
      weights[a-1][b-1] = weights[b-1][a-1] = c;
    }
    pair<int, VI> res = GetMinCut(weights);
    cout << "Case #" << i+1 << ": " << res.first << endl;
  }
}
// END CUT
```

4.9 Gomory

```
// Source: RR
// Tested with VOJ - MCQUERY

/*
 * Find min cut between every pair of vertices using N max_flow call (instead of N^2)
 * Not tested with directed graph
 * Index start from 0
 */
struct GomoryHu {
    int ok[MN], cap[MN][MN];
    int answer[MN][MN], parent[MN];
    int n;
    MaxFlow flow;

    GomoryHu(int n) : n(n), flow(n) {
        for(int i = 0; i < n; ++i) ok[i] = parent[i] = 0;
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j)
                cap[i][j] = 0, answer[i][j] = INF;
    }

    void addEdge(int u, int v, int c) {
        cap[u][v] += c; // An undirected edge must be added twice: (u,v) and (v,u)
    }

    void calc() {
        for(int i = 0; i < n; ++i) parent[i]=0;
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j)
                answer[i][j]=2000111000;

        for(int i = 1; i <= n-1; ++i) {
            flow = MaxFlow(n);
            REP(u,n) REP(v,n)
                if (cap[u][v])
                    flow.addEdge(u, v, cap[u][v]);

            int f = flow.getMaxFlow(i, parent[i]);

            bfs(i);
            for(int j = i+1; j < n; ++j)
                if (ok[j] && parent[j]==parent[i])
                    parent[j]=i;

            answer[i][parent[i]] = answer[parent[i]][i] = f;
            for(int j = 0; j < i; ++j)
                answer[i][j]=answer[j][i]=min(f,answer[parent[i]][j]);
        }
    }
    void bfs(int start) {
        memset(ok,0,sizeof ok);
        queue<int> qu;
        qu.push(start);
        while (!qu.empty()) {
            int u=qu.front(); qu.pop();
            for(int xid = 0; xid < flow.g[u].size(); ++xid) {
                int id = flow.g[u][xid];
                int v = flow.e[id].b, fl = flow.e[id].flow, cap = flow.e[id].cap;
                if (!ok[v] && fl < cap) {
                    ok[v]=1;
                    qu.push(v);
                }
            }
        }
    }
};
```

4.10 Stable Marrige

```
/*
```

```
 *      Takes a set of m men and n women, where each person has
 *      an integer preference for each of the persons of the opposite
 *      sex. Produces a matching of each man to some woman. The matching
 *      will have the following properties:
 *          - Each man is assigned a different woman (n must be at least m).
 *          - No two couples M1W1 and M2W2 will be unstable.
 *      Two couples are unstable if
 *          - M1 prefers W2 over W1 and
 *          - W1 prefers M2 over M1.
 * INPUTS:
 *   - m:        number of men.
 *   - n:        number of women (must be at least as large as m).
 *   - L[i][]:   the list of women in order of decreasing preference of man i.
 *   - R[j][i]:  the attractiveness of i to j.
 * OUTPUTS:
 *   - L2R[]:    the mate of man i (always between 0 and n-1)
 *   - R2L[]:    the mate of woman j (or -1 if single)
 * ALGORITHM:
 *   The algorithm is greedy and runs in time O(m^2).
 */

#define MAXM 1024
#define MAXW 1024
int m, n;
int L[MAXM][MAXW], R[MAXW][MAXM];
int L2R[MAXM], R2L[MAXW];
int p[MAXM];
void stableMarriage(){
    static int p[128];
    memset( R2L, -1, sizeof( R2L ) );
    memset( p, 0, sizeof( p ) );
    // Each man proposes...
    for( int i = 0; i < m; i++ ) {
        int man = i;
        while( man >= 0 ) {
            // to the next woman on his list in order of decreasing preference,
            // until one of them accepts;
            int wom;
            while( 1 ){
                wom = L[man][p[man]++];
                if( R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]] ) break;
            }
            // Remember the old husband of wom.
            int hubby = R2L[wom];
            // Marry man and wom.
            R2L[L2R[man] = wom] = man;
            // If a guy was dumped in the process, remarry him now.
            man = hubby;
        }
    }
}
```

4.11 Maximum Clique

```
class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
    int s[MV][MV/30+1];
    vector<int> sol;

    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }

    /* Zero Base */
    void addEdge(int u, int v) {
        if(u > v) swap(u, v);
        if(u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }

    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for(int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if(k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if(c == 0) {
            if(k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for(int i=0; i<(V+31)/32; i++) {
            for(int a = s[k][i]; a ; d++) {
                if(k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while(lb!=1) {
                    lb = (unsigned int)(lb) >> 1;
                    lg ++;
                }
                int u = i*32 + lg;
                if(k + dp[u] <= ans) return 0;
                if(dfs(u, k+1)) {
                    sol.push_back(v);
                    return 1;
                }
            }
        }
        return 0;
    }

    int solve() {
        for(int i=V-1; i>=0; i--) {
            dfs(i, 1);
            dp[i] = ans;
        }
        return ans;
    }
};
```

5 Math

5.1 Euclid

```
// This is a collection of useful code for solving problems that
// involve modular linear equations.  Note that all of the
// algorithms described here work on nonnegative integers.

typedef vector<int> VI;
typedef pair<int,int> PII;

int mod(int a, int b) { // return a % b (positive value)
    return ((a%b)+b)%b;
}

int gcd(int a, int b) { // computes gcd(a,b)
    int tmp;
    while(b){a%=b; tmp=a; a=b; b=tmp;}
    return a;
}

int lcm(int a, int b) { // computes lcm(a,b)
    return a/gcd(a,b)*b;
}

// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
```

```
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a/b;
        int t = b; b = a%b; a = t;
        t = xx; xx = x-q*xx; x = t;
        t = yy; yy = y-q*yy; y = t;
    }
    return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
    int x, y;
    VI solutions;
    int d = extended_euclid(a, n, x, y);
    if (!(b%d)) {
        x = mod (x*(b/d), n);
        for (int i = 0; i < d; i++)
            solutions.push_back(mod(x + i*(n/d), n));
    }
    return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
    int x, y;
    int d = extended_euclid(a, n, x, y);
    if (d > 1) return -1;
    return mod(x,n);
}

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
    int s, t;
    int d = extended_euclid(x, y, s, t);
    if (a%d != b%d) return make_pair(0, -1);
    return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
    PII ret = make_pair(a[0], x[0]);
    for (int i = 1; i < (int) x.size(); i++) {
        ret = chinese_remainder_theorem(ret.second, ret.first, x[i], a[i]);
        if (ret.second == -1) break;
    }
    return ret;
}

// computes x and y such that ax + by = c; on failure, x = y =-1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
    int d = gcd(a,b);
    if (c%d) {
        x = y = -1;
    } else {
        x = c/d * mod_inverse(a/d, b/d);
        y = (c-a*x)/b;
    }
}

int main() {
    cout << gcd(14, 30) << endl; // 2
    int x, y, d = extended_euclid(14, 30, x, y);
    cout << d << " " << x << " " << y << endl; // 2 -2 1
    VI sols = modular_linear_equation_solver(14, 30, 100); // 95 45
    for (int i = 0; i < (int) sols.size(); i++) cout << sols[i] << " ";
    cout << endl;
    cout << mod_inverse(8, 9) << endl; // 8
    int xs[] = {3, 5, 7, 4, 6};
    int as[] = {2, 3, 2, 3, 5};
    PII ret = chinese_remainder_theorem(VI (xs, xs+3), VI(as, as+3));
    cout << ret.first << " " << ret.second << endl; // 23 56
    ret = chinese_remainder_theorem (VI(xs+3, xs+5), VI(as+3, as+5));
    cout << ret.first << " " << ret.second << endl; // 11 12
    linear_diophantine(7, 2, 5, x, y);
    cout << x << " " << y << endl; // expected: 5 -15
}
```

5.2 Congruence

```
// Giai phuong trinh: a1x1 + a2x2 + ... + anxn = b (modul m)
// Trong do a1, a2, ..., an, b, m la cac so nguyen duong.

int g[MAXN], x[MAXN];

bool congruenceEquation(vector<int> a, int b, int m, vector<int> &ret) {
    int n = sz(a);
    a.pb(m);
    g[0] = a[0];
    For(i, 1, n) g[i] = gcd(g[i - 1], a[i]);
    ret.clear();
    if (b % g[n]) return false;
    int val = b / g[n];
    Ford(i, n, 1) {
        pair<ll, ll> p = extgcd(g[i - 1], a[i]);
        x[i] = p.se * val % m;
        val = p.fi * val % m;
    }
    x[0] = val;
    For(i, 0, n) x[i] = (x[i] + m) % m;
    Rep(i, n) ret.pb(x[i]);
    return true;
}
```

5.3 Gaussian

```
// Gauss-Jordan elimination.
// Returns: number of solution (0, 1 or INF)
//   When the system has at least one solution, ans will contains
//   one possible solution
// Possible improvement when having precision errors:
//   - Divide i-th row by a(i, i)
//   - Choosing pivoting row with min absolute value (sometimes this is better that
    maximum, as implemented here)
// Tested:
// - https://open.kattis.com/problems/equationsolver
// - https://open.kattis.com/problems/equationsolverplus
int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }
```

```
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    // If we need any solution (in case INF solutions), we should be
    // ok at this point.
    // If need to solve partially (get which values are fixed/INF value):
//    for (int i=0; i<m; ++i)
//        if (where[i] != -1) {
//            REP(j,n) if (j != i && fabs(a[where[i]][j]) > EPS) {
//                where[i] = -1; break;
//            }
//        }
    // Then the variables which has where[i] == -1 --> INF values

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

5.4 Phi

```
int eulerPhi(int n) { // = n (1-1/p1) ... (1-1/pn)
    if (n == 0) return 0;
    int ans = n;
    for (int x = 2; x*x <= n; ++x) {
        if (n % x == 0) {
            ans -= ans / x;
            while (n % x == 0) n /= x;
        }
    }
    if (n > 1) ans -= ans / n;
    return ans;
}
// LookUp Version
const int N = 1000000;
int eulerPhi(int n) {
    static int lookup = 0, p[N], f[N];
    if (!lookup) {
        REP(i,N) p[i] = 1, f[i] = i;
        for (int i = 2; i < N; ++i) {
            if (p[i]) {
            f[i] -= f[i] / i;
                for (int j = i+i; j < N; j+=i)
                    p[j] = 0, f[j] -= f[j] / i;
            }
        }
        lookup = 1;
    }
    return f[n];
}
```

5.5 Rabin Miller

```
bool suspect(ll a, ll s, ll d, ll n) {
   ll x = powMod(a, d, n);
   if (x == 1) return true;
   for (int r = 0; r < s; ++r) {
      if (x == n - 1) return true;
      x = mulMod(x, x, n);
   }
   return false;
}
// {2,7,61,-1}                 is for n < 4759123141 (= 2^32)
// {2,3,5,7,11,13,17,19,23,-1} is for n < 10^16 (at least)
bool isPrime(ll n) {
   if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
   ll test[] = {2,3,5,7,11,13,17,19,23,-1};
   ll d = n - 1, s = 0;
   while (d % 2 == 0) ++s, d /= 2;
   for (int i = 0; test[i] < n && test[i] != -1; ++i)
      if (!suspect(test[i], s, d, n)) return false;
   return true;
}

// Killer prime: 5555555557LL (fail when not used mulMod)
```

6 String

6.1 Aho Corasick

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

using namespace std;

const int MAXLEN = 1000005;
const int MAXN = 100005;

// Trie
struct Node {
        int ch[26];
        Node() {
                memset(ch, -1, sizeof ch);
        }
} trie[MAXLEN]; int sz;

// Aho Corasick
struct AhoCorasick {
        int fail[MAXLEN];
        vector <int> g[MAXLEN];

        void add(string &s, int id) {
                int cur = 0;
                for (int i = 0; i < s.size(); ++i) {
                        int c = s[i] - 'a';
                        if (trie[cur].ch[c] == 0) {
                                trie[cur].ch[c] = ++sz;
                        }
                        cur = trie[cur].ch[c];
                }
        }

        void bfs() {
                queue <int> q; q.push(0);
                while(!q.empty()) {
                        int u = q.front(); q.pop();
                        for (int i = 0; i < 26; ++i) {
                                int v = trie[u].ch[i];
                                trie[u].ch[i] = 0;
                                trie[u].ch[i] = trie[fail[u]].ch[i];
                                if (v) {
                                        fail[v] = trie[u].ch[i];
                                        trie[u].ch[i] = v;
                                        q.push(v);
                                }
                        }
                }
        }

        void build() {
                // build link-tree
                for (int i = 1; i <= sz; ++i) {
                        g[fail[i]].push_back(i);
                }
                // dfs(0);
```

```
    }
};
```

6.2 Manacher

```
const char DUMMY = '.';

int manacher(string s) {
    // Add dummy character to not consider odd/even length
    // NOTE: Ensure DUMMY does not appear in input
    // NOTE: Remember to ignore DUMMY when tracing

    int n = s.size() * 2 - 1;
    vector <int> f = vector <int>(n, 0);
    string a = string(n, DUMMY);
    for (int i = 0; i < n; i += 2) a[i] = s[i / 2];

    int l = 0, r = -1, center, res = 0;
    for (int i = 0, j = 0; i < n; i++) {
        j = (i > r ? 0 : min(f[l + r - i], r - i)) + 1;
        while (i - j >= 0 && i + j < n && a[i - j] == a[i + j]) j++;
        f[i] = --j;
        if (i + j > r) {
            r = i + j;
            l = i - j;
        }

        int len = (f[i] + i % 2) / 2 * 2 + 1 - i % 2;
        if (len > res) {
            res = len;
            center = i;
        }
    }
    // a[center - f[center]..center + f[center]] is the needed substring
    return res;
}
```

6.3 Lexicographically minimal rotated string

```
// Tinh vi tri cua xau xoay vong co thu tu tu dien nho nhat cua xau s[]
int minmove(string s) {
    int n = s.length();
    int x, y, i, j, u, v; // x is the smallest string before string y
    for (x = 0, y = 1; y < n; ++ y) {
        i = u = x;
        j = v = y;
        while (s[i] == s[j]) {
            ++ u; ++ v;
            if (++ i == n) i = 0;
            if (++ j == n) j = 0;
            if (i == x) break; // All strings are equal
        }
        if (s[i] <= s[j]) y = v;
        else {
            x = y;
            if (u > y) y = u;
        }
    }
    return x;
}
```

6.4 Z Function

```
vector<int> calcZ(const string &s) {
    int n = s.size();
    vector<int> z (n);
    for (int i = 1, j = 0; i < n; ++i) {
        if (j + z[j] > i) z[i] = min(j + z[j] - i, z[i - j]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (j + z[j] <= i || i + z[i] > j + z[j]) j = i;
    }
    return z;
}
```

7 Geometry

7.1 Basic (Point, Line)

```
#define EPS 1e-6
const double PI = acos(-1.0);

double DEG_to_RAD(double d) { return d * PI / 180.0; }
double RAD_to_DEG(double r) { return r * 180.0 / PI; }

inline int cmp(double a, double b) {
    return (a < b - EPS) ? -1 : ((a > b + EPS) ? 1 : 0);
}

struct Point {
    double x, y;
    Point() { x = y = 0.0; }
    Point(double x, double y) : x(x), y(y) {}

    Point operator + (const Point& a) const { return Point(x+a.x, y+a.y); }
    Point operator - (const Point& a) const { return Point(x-a.x, y-a.y); }
    Point operator * (double k) const { return Point(x*k, y*k); }
    Point operator / (double k) const { return Point(x/k, y/k); }

    double operator * (const Point& a) const { return x*a.x + y*a.y; } // dot product
    double operator % (const Point& a) const { return x*a.y - y*a.x; } // cross
         product

    int cmp(Point q) const { if (int t = ::cmp(x,q.x)) return t; return ::cmp(y,q.y);
         }

    #define Comp(x) bool operator x (Point q) const { return cmp(q) x 0; }
    Comp(>) Comp(<) Comp(==) Comp(>=) Comp(<=) Comp(!=)
    #undef Comp

    Point conj() { return Point(x, -y); }
    double norm() { return x*x + y*y; }

    // Note: There are 2 ways for implementing len():
    // 1. sqrt(norm()) --> fast, but inaccurate (produce some values that are of order
         X^2)
    // 2. hypot(x, y) --> slow, but much more accurate
    double len() { return sqrt(norm()); }

    Point rotate(double alpha) {
        double cosa = cos(alpha), sina = sin(alpha);
        return Point(x * cosa - y * sina, x * sina + y * cosa);
    }
};

int ccw(Point a, Point b, Point c) {
    return cmp((b-a)%(c-a),0);
}
int RE_TRAI = ccw(Point(0, 0), Point(0, 1), Point(-1, 1));
int RE_PHAI = ccw(Point(0, 0), Point(0, 1), Point(1, 1));
istream& operator >> (istream& cin, Point& p) {
    cin >> p.x >> p.y;
    return cin;
}
ostream& operator << (ostream& cout, Point& p) {
    cout << p.x << ' ' << p.y;
    return cout;
}

double angle(Point a, Point o, Point b) { // min of directed angle AOB & BOA
    a = a - o; b = b - o;
    return acos((a * b) / sqrt(a.norm()) / sqrt(b.norm()));
}
```

```
double directed_angle(Point a, Point o, Point b) { // angle AOB, in range [0, 2*PI)
    double t = -atan2(a.y - o.y, a.x - o.x)
            + atan2(b.y - o.y, b.x - o.x);
    while (t < 0) t += 2*PI;
    return t;
}

// Distance from p to Line ab (closest Point --> c)
double distToLine(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    c = a + (ab * u);
    return (p-c).len();
}

// Distance from p to segment ab (closest Point --> c)
double distToLineSegment(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    if (u < 0.0) {
        c = Point(a.x, a.y);
        return (p - a).len();
    }
    if (u > 1.0) {
        c = Point(b.x, b.y);
        return (p - b).len();
    }
    return distToLine(p, a, b, c);
}

// NOTE: WILL NOT WORK WHEN a = b = 0.
struct Line {
    double a, b, c;
    Point A, B; // Added for polygon intersect line. Do not rely on assumption that
         these are valid

    Line(double a, double b, double c) : a(a), b(b), c(c) {}

    Line(Point A, Point B) : A(A), B(B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = - (a * A.x + b * A.y);
    }
    Line(Point P, double m) {
        a = -m; b = 1;
        c = -((a * P.x) + (b * P.y));
    }
    double f(Point A) {
        return a*A.x + b*A.y + c;
    }
};

bool areParallel(Line l1, Line l2) {
    return cmp(l1.a*l2.b, l1.b*l2.a) == 0;
}

bool areSame(Line l1, Line l2) {
    return areParallel(l1 ,l2) && cmp(l1.c*l2.a, l2.c*l1.a) == 0
                && cmp(l1.c*l2.b, l1.b*l2.c) == 0;
}

bool areIntersect(Line l1, Line l2, Point &p) {
    if (areParallel(l1, l2)) return false;
    double dx = l1.b*l2.c - l2.b*l1.c;
    double dy = l1.c*l2.a - l2.c*l1.a;
    double d  = l1.a*l2.b - l2.a*l1.b;
    p = Point(dx/d, dy/d);
    return true;
}

void closestPoint(Line l, Point p, Point &ans) {
    if (fabs(l.b) < EPS) {
        ans.x = -(l.c) / l.a; ans.y = p.y;
        return;
    }
    if (fabs(l.a) < EPS) {
        ans.x = p.x; ans.y = -(l.c) / l.b;
        return;
    }
    Line perp(l.b, -l.a, - (l.b*p.x - l.a*p.y));
    areIntersect(l, perp, ans);
}

void reflectionPoint(Line l, Point p, Point &ans) {
    Point b;
    closestPoint(l, p, b);
    ans = p + (b - p) * 2;
}
```

7.2 Polygon

```
typedef vector< Point > Polygon;

// Convex Hull:
// If minimum point --> #define REMOVE_REDUNDANT
// If maximum point --> need to change >= and <= to > and < (see Note).
// Known issues:
// - Max. point does not work when some points are the same.
// Tested:
// - https://open.kattis.com/problems/convexhull
/*
bool operator<(const Point &rhs) const { return make_pair(y,x) < make_pair(rhs.y,rhs.x
     ); }
bool operator==(const Point &rhs) const { return make_pair(y,x) == make_pair(rhs.y,rhs
     .x); }
 */
double area2(Point a, Point b, Point c) { return a%b + b%c + c%a; }
#ifdef REMOVE_REDUNDANT
bool between(const Point &a, const Point &b, const Point &c) {
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y
         ) <= 0);
}
#endif

void ConvexHull(vector<Point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<Point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        // Note: If need maximum points on convex hull, need to change >= and <= to >
             and <.
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.
             pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.
             pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
#endif
}

// Area, perimeter, centroid
double signed_area(Polygon p) {
    double area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
```

```
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}
double area(const Polygon &p) {
    return fabs(signed_area(p));
}
Point centroid(Polygon p) {
    Point c(0,0);
    double scale = 6.0 * signed_area(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
}
double perimeter(Polygon P) {
    double res = 0;
    for(int i = 0; i < P.size(); ++i) {
        int j = (i + 1) % P.size();
        res += (P[i] - P[j]).len();
    }
    return res;
}
// Is convex: checks if polygon is convex. Assume there are no 3 collinear points
bool is_convex(const Polygon &P) {
    int sz = (int) P.size();
    if (sz <= 2) return false;
    int isLeft = ccw(P[0], P[1], P[2]);
    for (int i = 1; i < sz; i++)
        if (ccw(P[i], P[(i+1) % sz], P[(i+2) % sz]) * isLeft < 0)
            return false;
    return true;
}

// Inside polygon: O(N). Works with any polygon
// Tested:
// - https://open.kattis.com/problems/pointinpolygon
// - https://open.kattis.com/problems/cuttingpolygon
bool in_polygon(const Polygon &p, Point q) {
    if ((int)p.size() == 0) return false;

    // Check if point is on edge.
    int n = SZ(p);
    REP(i,n) {
        int j = (i + 1) % n;
        Point u = p[i], v = p[j];

        if (u > v) swap(u, v);

        if (ccw(u, v, q) == 0 && u <= q && q <= v) return true;
    }

    // Check if point is strictly inside.
    int c = 0;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        if ((p[i].y <= q.y && q.y < p[j].y || p[j].y <= q.y && q.y < p[i].y) && q.x <
             p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y)) c = !c;
    }
    return c;
}

// Check point in convex polygon, O(logN)
// Source: http://codeforces.com/contest/166/submission/1392387
// On edge --> false
#define Det(a,b,c) ((double)(b.x-a.x)*(double)(c.y-a.y)-(double)(b.y-a.y)*(c.x-a.x))
bool in_convex(vector<Point>& l, Point p){
    int a = 1, b = l.size()-1, c;
    if (Det(l[0], l[a], l[b]) > 0) swap(a,b);
    // Allow on edge --> if (Det... > 0 || Det ... < 0)
    if (Det(l[0], l[a], p) >= 0 || Det(l[0], l[b], p) <= 0) return false;
    while(abs(a-b) > 1) {
        c = (a+b)/2;
        if (Det(l[0], l[c], p) > 0) b = c; else a = c;
    }
    // Alow on edge --> return Det... <= 0
    return Det(l[a], l[b], p) < 0;
}
```

```
// Cut a polygon with a line. Returns one half.
// To return the other half, reverse the direction of Line l (by negating l.a, l.b)
// The line must be formed using 2 points
Polygon polygon_cut(const Polygon& P, Line l) {
    Polygon Q;
    for(int i = 0; i < P.size(); ++i) {
        Point A = P[i], B = (i == P.size()-1) ? P[0] : P[i+1];
        if (ccw(l.A, l.B, A) != -1) Q.push_back(A);
        if (ccw(l.A, l.B, A)*ccw(l.A, l.B, B) < 0) {
            Point p; areIntersect(Line(A, B), l, p);
            Q.push_back(p);
        }
    }
    return Q;
}

// Find intersection of 2 convex polygons
// Helper method
bool intersect_1pt(Point a, Point b,
    Point c, Point d, Point &r) {
    double D =  (b - a) % (d - c);
    if (cmp(D, 0) == 0) return false;
    double t =  ((c - a) % (d - c)) / D;
    double s = -((a - c) % (b - a)) / D;
    r = a + (b - a) * t;
    return cmp(t, 0) >= 0 && cmp(t, 1) <= 0 && cmp(s, 0) >= 0 && cmp(s, 1) <= 0;
}
Polygon convex_intersect(Polygon P, Polygon Q) {
    const int n = P.size(), m = Q.size();
    int a = 0, b = 0, aa = 0, ba = 0;
    enum { Pin, Qin, Unknown } in = Unknown;
    Polygon R;
    do {
        int a1 = (a+n-1) % n, b1 = (b+m-1) % m;
        double C = (P[a] - P[a1]) % (Q[b] - Q[b1]);
        double A = (P[a1] - Q[b]) % (P[a] - Q[b]);
        double B = (Q[b1] - P[a]) % (Q[b] - P[a]);
        Point r;
        if (intersect_1pt(P[a1], P[a], Q[b1], Q[b], r)) {
            if (in == Unknown) aa = ba = 0;
            R.push_back( r );
            in = B > 0 ? Pin : A > 0 ? Qin : in;
        }
        if (C == 0 && B == 0 && A == 0) {
            if (in == Pin) { b = (b + 1) % m; ++ba; }
            else            { a = (a + 1) % m; ++aa; }
        } else if (C >= 0) {
            if (A > 0) { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
            else       { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
        } else {
            if (B > 0) { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
            else       { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
        }
    } while ( (aa < n || ba < m) && aa < 2*n && ba < 2*m );
    if (in == Unknown) {
        if (in_convex(Q, P[0])) return P;
        if (in_convex(P, Q[0])) return Q;
    }
    return R;
}

// Find the diameter of polygon.
// Rotating callipers
double convex_diameter(Polygon pt) {
    const int n = pt.size();
    int is = 0, js = 0;
    for (int i = 1; i < n; ++i) {
        if (pt[i].y > pt[is].y) is = i;
        if (pt[i].y < pt[js].y) js = i;
    }
    double maxd = (pt[is]-pt[js]).norm();
    int i, maxi, j, maxj;
    i = maxi = is;
    j = maxj = js;
    do {
        int jj = j+1; if (jj == n) jj = 0;
        if ((pt[i] - pt[jj]).norm() > (pt[i] - pt[j]).norm()) j = (j+1) % n;
```

```
        else i = (i+1) % n;
        if ((pt[i]-pt[j]).norm() > maxd) {
            maxd = (pt[i]-pt[j]).norm();
            maxi = i; maxj = j;
        }
    } while (i != is || j != js);
    return maxd; /* farthest pair is (maxi, maxj). */
}
/*
------- True Method (from AI.Cash) ---------
template <class F>
F maxDist2(const Polygon<F>& poly) {
  int n = static_cast<int>(poly.size());
  F res = F(0);
  for (int i = 0, j = n < 2 ? 0 : 1; i < j; ++i)
    for (;; j = next(j, n)) {
      res = max(res, dist2(poly[i], poly[j]));
      if (ccw(poly[i+1] - poly[i], poly[next(j, n)] - poly[j]) >= 0) break;
    }
  return res;
}
*/

// Closest pair
// Source: e-maxx.ru
// Tested:
// - https://open.kattis.com/problems/closestpair2
// - https://open.kattis.com/problems/closestpair1
// Notes:
// - Sort by X first
// - Implement compare by Y
#define upd_ans(x, y) {}
#define MAXN 100
double mindist = 1e20; // will be the result
void rec(int l, int r, Point a[]) {
    if (r - l <= 3) {
        for (int i=l; i<=r; ++i)
            for (int j=i+1; j<=r; ++j)
                    upd_ans(a[i], a[j]);
        sort(a+l, a+r+1, cmpy); // compare by y
        return;
    }

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec(l, m, a), rec(m+1, r, a);
    static Point t[MAXN];
    merge(a+l, a+m+1, a+m+1, a+r+1, t, cmpy); // compare by y
    copy(t, t+r-l+1, a+l);

    int tsz = 0;
    for (int i=l; i<=r; ++i)
        if (fabs(a[i].x - midx) < mindist) {
            for (int j=tsz-1; j>=0 && a[i].y - t[j].y < mindist; --j)
                upd_ans(a[i], t[j]);
            t[tsz++] = a[i];
        }
}

// Pick theorem
// Given non-intersecting polygon.
// S = area
// I = number of integer points strictly Inside
// B = number of points on sides of polygon
// S = I + B/2 - 1

// Check if we can form triangle with edges x, y, z.
bool isSquare(long long x) { /* */ }
bool isIntegerCoordinates(int x, int y, int z) {
    long long s=(long long)(x+y+z)*(x+y-z)*(x+z-y)*(y+z-x);
    return (s%4==0 && isSquare(s/4));
}
```

7.3 Circle

```
struct Circle : Point {
    double r;
    Circle(double x = 0, double y = 0, double r = 0) : Point(x, y), r(r) {}
    Circle(Point p, double r) : Point(p), r(r) {}

    bool contains(Point p) { return (*this - p).len() <= r + EPS; }
};

// Find common tangents to 2 circles
// Tested:
// - http://codeforces.com/gym/100803/ - H
// Helper method
void tangents(Point c, double r1, double r2, vector<Line> & ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -EPS)  return;
    d = sqrt(fabs(d));
    Line l((c.x * r + c.y * d) / z,
            (c.y * r - c.x * d) / z,
            r1);
    ans.push_back(l);
}
// Actual method: returns vector containing all common tangents
vector<Line> tangents(Circle a, Circle b) {
    vector<Line> ans; ans.clear();
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents(b-a, a.r*i, b.r*j, ans);
    for(int i = 0; i < ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;

    vector<Line> ret;
    for(int i = 0; i < (int) ans.size(); ++i) {
        bool ok = true;
        for(int j = 0; j < i; ++j)
            if (areSame(ret[j], ans[i])) {
                ok = false;
                break;
            }
        if (ok) ret.push_back(ans[i]);
    }
    return ret;
}

// Circle & line intersection
// Tested:
// - http://codeforces.com/gym/100803/ - H
vector<Point> intersection(Line l, Circle cir) {
    double r = cir.r, a = l.a, b = l.b, c = l.c + l.a*cir.x + l.b*cir.y;
    vector<Point> res;

    double x0 = -a*c/(a*a+b*b),  y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS) return res;
    else if (fabs(c*c - r*r*(a*a+b*b)) < EPS) {
        res.push_back(Point(x0, y0) + Point(cir.x, cir.y));
        return res;
    }
    else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax,ay,bx,by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;

        res.push_back(Point(ax, ay) + Point(cir.x, cir.y));
        res.push_back(Point(bx, by) + Point(cir.x, cir.y));
        return res;
    }
}

// helper functions for commonCircleArea
double cir_area_solve(double a, double b, double c) {
    return acos((a*a + b*b - c*c) / 2 / a / b);
}
double cir_area_cut(double a, double r) {
    double s1 = a * r * r / 2;
    double s2 = sin(a) * r * r / 2;
```

```
        return s1 - s2;
    }
    // Tested: http://codeforces.com/contest/600/problem/D
    double commonCircleArea(Circle c1, Circle c2) { //return the common area of two circle
        if (c1.r < c2.r) swap(c1, c2);
        double d = (c1 - c2).len();
        if (d + c2.r <= c1.r + EPS) return c2.r*c2.r*M_PI;
        if (d >= c1.r + c2.r - EPS) return 0.0;
        double a1 = cir_area_solve(d, c1.r, c2.r);
        double a2 = cir_area_solve(d, c2.r, c1.r);
        return cir_area_cut(a1*2, c1.r) + cir_area_cut(a2*2, c2.r);
    }

    // Check if 2 circle intersects. Return true if 2 circles touch
    bool areIntersect(Circle u, Circle v) {
        if (cmp((u - v).len(), u.r + v.r) > 0) return false;
        if (cmp((u - v).len() + v.r, u.r) < 0) return false;
        if (cmp((u - v).len() + u.r, v.r) < 0) return false;
        return true;
    }

    // If 2 circle touches, will return 2 (same) points
    // If 2 circle are same --> be careful
    // Tested:
    // - http://codeforces.com/gym/100803/ - H
    // - http://codeforces.com/gym/100820/ - I
    vector<Point> circleIntersect(Circle u, Circle v) {
        vector<Point> res;
        if (!areIntersect(u, v)) return res;
        double d = (u - v).len();
        double alpha = acos((u.r * u.r + d*d - v.r * v.r) / 2.0 / u.r / d);

        Point p1 = (v - u).rotate(alpha);
        Point p2 = (v - u).rotate(-alpha);
        res.push_back(p1 / p1.len() * u.r + u);
        res.push_back(p2 / p2.len() * u.r + u);
        return res;
    }
```

7.4 Smallest Enclosing Circle

```
    // Smallest enclosing circle:
    // Given N points. Find the smallest circle enclosing these points.
    // Amortized complexity: O(N)

    struct SmallestEnclosingCircle {
        Circle getCircle(vector<Point> points) {
            assert(!points.empty());

            random_shuffle(points.begin(), points.end());
            Circle c(points[0], 0);
            int n = points.size();

            for (int i = 1; i < n; i++)
                if ((points[i] - c).len() > c.r + EPS)
                {
                    c = Circle(points[i], 0);
                    for (int j = 0; j < i; j++)
                        if ((points[j] - c).len() > c.r + EPS)
                        {
                            c = Circle((points[i] + points[j]) / 2, (points[i] - points[j
                                ]).len() / 2);
                            for (int k = 0; k < j; k++)
                                if ((points[k] - c).len() > c.r + EPS)
                                    c = getCircumcircle(points[i], points[j], points[k]);
                        }
                }

            return c;
        }

        // NOTE: This code work only when a, b, c are not collinear and no 2 points are
            same --> DO NOT
        // copy and use in other cases.
        Circle getCircumcircle(Point a, Point b, Point c) {
            assert(a != b && b != c && a != c);
            assert(ccw(a, b, c));

            double d = 2.0 * (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y));
            assert(fabs(d) > EPS);
            double x = (a.norm() * (b.y - c.y) + b.norm() * (c.y - a.y) + c.norm() * (a.y
                - b.y)) / d;
            double y = (a.norm() * (c.x - b.x) + b.norm() * (a.x - c.x) + c.norm() * (b.x
                - a.x)) / d;
            Point p(x, y);
            return Circle(p, (p - a).len());
        }
    };
```

8 Misc

8.1 Lagrange polynomial

Given a set of $k+1$ data points $(x_0, y_0), ..., (x_j, y_j), ..., (x_k, y_k)$ where no two x_j are the same, the interpolation polynomial in the Lagrange form is a linear combination

$L(x) := \sum_{j=0}^{k} y_j \ell_j(x)$ of Lagrange basis polynomials

$\ell_j(x) := \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} = \frac{(x-x_0)}{(x_j-x_0)} \cdots \frac{(x-x_{j-1})}{(x_j-x_{j-1})} \frac{(x-x_{j+1})}{(x_j-x_{j+1})} \cdots \frac{(x-x_k)}{(x_j-x_k)}$

8.2 Planar graph

Theorem 1. $e \le 3v - 6$.
Theorem 2. If there are no cycles of length 3, then $e \le 2v - 4$.
Theorem 3. $f \le 2v - 4$.
Euler's formula: $v - e + f = 2$.

8.3 Catalan number

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n}; C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} = \frac{2(2n+1)}{n+2} C_n$$

$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440$

8.4 Stirling number

* **First kind: number of permutations of n numbers with k cycles.**

$\left[{n+1 \atop k}\right] = n\left[{n \atop k}\right] + \left[{n \atop k-1}\right]$

for $k > 0$, with the initial conditions $\left[{0 \atop 0}\right] = 1$ and $\left[{n \atop 0}\right] = \left[{0 \atop n}\right] = 0$ for $n > 0$.

We have: $(x)^{(n)} = x(x+1)\cdots(x+n-1) = \sum_{k=0}^{n} \left[{n \atop k}\right] x^k$

Signed Stirling number of the first kind: $s(n,k) = (-1)^{n-k}\left[{n \atop k}\right]$, $s(n+1,k) = -ns(n,k) + s(n,k-1)$.

It's also given that: $(x)_n = x(x-1)(x-2)\cdots(x-n+1) = \sum_{k=0}^{n} s(n,k)x^k$.

-	k=0	k=1	k=2	k=3	k=4	k=5	k=6
n=0	1	-	-	-	-	-	-
n=1	0	1	-	-	-	-	-
n=2	0	1	1	-	-	-	-
n=3	0	2	3	1	-	-	-
n=4	0	6	11	6	1	-	-
n=5	0	24	50	35	10	1	-
n=6	0	120	274	225	85	15	1

* **Second kind: number of ways to partition a set of n objects into k non-empty subsets and is denoted by $S(n,k)$ or $\left\{ {n \atop k} \right\}$.**

Explicit formula: $\left\{ {n \atop k} \right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$

$\left\{ {n+1 \atop k} \right\} = k\left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$

Related recurrences

1. $\sum_{k=0}^{n}\left\{ {n \atop k} \right\}(x)_k = x^n$
2. $\left\{ {n+1 \atop k+1} \right\} = \sum_{j=k}^{n}\binom{n}{j}\left\{ {j \atop k} \right\}$
3. $\left\{ {n+1 \atop k+1} \right\} = \sum_{j=k}^{n}(k+1)^{n-j}\left\{ {j \atop k} \right\}$
4. $\left\{ {n+k+1 \atop k} \right\} = \sum_{j=0}^{k} j\left\{ {n+j \atop j} \right\}$

-	k=0	k=1	k=2	k=3	k=4	k=5	k=6
n=0	1	-	-	-	-	-	-
n=1	0	1	-	-	-	-	-
n=2	0	1	1	-	-	-	-
n=3	0	1	3	1	-	-	-
n=4	0	1	7	6	1	-	-
n=5	0	1	15	25	10	1	-
n=6	0	1	31	90	65	15	1

Variants

1. ***Associated Stirling numbers of the second kind:*** An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n,k)$ and obeys the recurrence relation
$S_r(n+1,k) = k\ S_r(n,k) + \binom{n}{r-1}S_r(n-r+1,k-1)$
2. ***Reduced Stirling numbers of the second kind:*** Define the reduced Stirling numbers of the second kind, denoted $S^d(n,k)$, to be the number of ways to partition the integers 1, 2, ..., n into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i-j| \geq d$. It has been shown that these numbers satisfy
$S^d(n,k) = S(n-d+1,k-d+1), n \geq k \geq d$

8.5 Bell number

Definition: B_n ***is the number of partitions of a set of size*** n***.*** A partition of a set S is defined as a set of nonempty, pairwise disjoint subsets of S whose union is S.
B = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, 1382958545, 10480142147, 82864869804, 682076806159, 5832742205057, ...
Formula:
$B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k,\ B_n = \sum_{k=0}^{n}\left\{ {n \atop k} \right\}$
$B_{n+m} = \sum_{k=0}^{n}\sum_{j=0}^{m}\left\{ {m \atop j} \right\}\binom{n}{k}j^{n-k}B_k$. [Spidey (2008)]

8.6 Eulerian number

Definition: the Eulerian number $A(n,m)$ is the number of permutations of the numbers 1 to n in which exactly m elements are greater than the previous element (permutations with m "ascents")
Basic properties
$A(n,0) = A(n,n-1) = 1$ for all values of n.
$A(n,m) = (n-m)A(n-1,m-1) + (m+1)A(n-1,m)$
Explicit formula: $A(n,m) = \sum_{k=0}^{m}(-1)^k\binom{n+1}{k}(m+1-k)^n$.

-	k=0	k=1	k=2	k=3	k=4	k=5	k=6
n=1	1	-	-	-	-	-	-
n=2	1	1	-	-	-	-	-
n=3	1	4	1	-	-	-	-
n=4	1	11	11	1	-	-	-
n=5	1	26	66	26	1	-	-
n=6	1	57	302	302	57	1	-
n=7	1	120	1191	2416	1191	120	1

Eulerian number of the 2nd kind: The permutations of the multiset $\{1,1,2,2,...,n,n\}$ which have the property that for each k, all the numbers appearing between the two occurrences of k in the permutation are greater than k are counted by the double factorial number $(2n-1)!!$. The Eulerian number of the second kind, denoted $\left\langle\!\left\langle {n \atop m} \right\rangle\!\right\rangle$, counts the number of all such permutations that have exactly m ascents.
The Eulerian numbers of the second kind satisfy the recurrence relation, that follows directly from the above definition:
$\left\langle\!\left\langle {n \atop m} \right\rangle\!\right\rangle = (2n-m-1)\left\langle\!\left\langle {n-1 \atop m-1} \right\rangle\!\right\rangle + (m+1)\left\langle\!\left\langle {n-1 \atop m} \right\rangle\!\right\rangle$,
with initial condition for n = 0, expressed in Iverson bracket notation:
$\left\langle\!\left\langle {0 \atop m} \right\rangle\!\right\rangle = [m=0]$.

8.7 Combinatorics

1. $\binom{n}{k} = \binom{n}{n-k}$
2. $\binom{n}{k+1} = \binom{n-1}{k+1} + \binom{n-1}{k}$
3. $k\binom{n}{k} = n\binom{n-1}{k-1}$
4. $k\binom{n}{k} = (n-k+1)\binom{n}{k-1}$
5. $\binom{n}{0} + \binom{n+1}{1} + \binom{n+2}{2} + ... + \binom{n+k}{k} = \binom{n+k+1}{k}$
6. $\binom{n}{n} + \binom{n+1}{n} + \binom{n+2}{n} + ... + \binom{n+k}{n} = \binom{n+k+1}{n+1}$
7. $\binom{n}{0} - \binom{n}{1} + \binom{n}{2} - ... + (-1)^n\binom{n}{n} = 0$
8. $\binom{n}{1} + 2\binom{n}{2} + 3\binom{n}{3} + ... + n\binom{n}{n} = n2^{n-1}$
9. $\binom{n}{k}$ is divisible by n if n is prime and $1 \leq k \leq n-1$
10. [Vandermonde] $\binom{m+n}{k} = \sum_{i=0}^{k}\binom{m}{i}\binom{n}{k-i}$

8.8 Euler's totient function Lemma

For all n and m, and $e \geq log_2(m)$ it holds that:
n^e % m = $n^{\phi(m)+e\%\phi(m)}$ % m.

8.9 Burnside's Lemma

Let G be a finite group that acts on a set X. For each g in G let X^g denote the set of elements in X^g that are fixed by g (also said to be left invariant by g), i.e. $X^g = \{x \in X | g.x = x\}$. Burnside's lemma asserts the following formula for the number of orbits, denoted $|X/G|$:
$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|$.

8.10 Johnson's rule for schedualing jobs

2 machines:
- Job i: A[i], B[i]
- Sort by min(A[i], B[i])
- Consider job i (sorted order):
if A[i] $\leq$ B[i] $\rightarrow$ schedule first
else schedule last

8.11 Dilworth's theorem

- An antichain in a partially ordered set is a set of elements no two of which are comparable to each other.
- A chain is a set of elements every two of which are comparable.
- In any finite partially ordered set, the maximum number of elements in any antichain equals the minimum number of chains in any partition of the set into chains.

8.12 Properties of Hamiltonian Path

1. All Hamiltonian graphs are biconnected, but a biconnected graph need not be Hamiltonian.
2. The number of different Hamiltonian cycles in a complete undirected graph on n vertices is $(n-1)!/2$ and in a complete directed graph on n vertices is $(n-1)!$. These counts assume that cycles that are the same apart from their starting point are not counted separately.

BondyChvtal theorem: BondyChvtal theorem operates on the closure $cl(G)$ of a graph G with n vertices, obtained by repeatedly adding a new edge uv connecting a nonadjacent pair of vertices u and v with $degree(v) + degree(u) \leq n$ until no more pairs with this property can be found.
A graph is Hamiltonian if and only if its closure is Hamiltonian.

Dirac (1952): A simple graph with n vertices ($n \geq 3$) is Hamiltonian if every vertex has degree $n/2$ or greater.

Ore (1960): A graph with n vertices ($n \geq 3$) is Hamiltonian if, for every pair of non-adjacent vertices, the sum of their degrees is n or greater.

Ghouila-Houiri (1960): A strongly connected simple directed graph with n vertices is Hamiltonian if every vertex has a full degree greater than or equal to n.

Meyniel (1973): A strongly connected simple directed graph with n vertices is Hamiltonian if the sum of full degrees of every pair of distinct non-adjacent vertices is greater than or equal to $2n-1$.

8.13 Geometry Formulas

Given a triangle ABC, we have several formulas:

1. Compute the length of median lines:
$m_a^2 = \frac{2(b^2+c^2)-a^2}{4}; m_b^2 = \frac{2(a^2+c^2)-b^2}{4}; m_c^2 = \frac{2(a^2+b^2)-c^2}{4}$
2. Compute the length of bisectors:
$l_a^2 = \frac{bc}{(b+c)^2}\left[(b+c)^2 - a^2\right];$
$l_b^2 = \frac{ac}{(a+c)^2}\left[(a+c)^2 - b^2\right];$
$l_c^2 = \frac{ab}{(a+b)^2}\left[(a+b)^2 - c^2\right];$
3. Assume the area of ABC is S, it holds that:
$S = \frac{1}{2}ah_a; S = \sqrt{p(p-a)(p-b)(p-c)}; S = \frac{abc}{4R}; S = pr;$
$S = (p-a)R_a$
4. Law of sines
$\frac{a}{sinA} = \frac{b}{sinB} = \frac{c}{sinC} = 2R$
5. Law of cosines:
$c^2 = a^2 + b^2 - 2abcosC$

Sphere: $V = \frac{4}{3}\pi r^3; A = 4\pi r^2$

$V = \frac{\pi h}{6}\left(3a^2 + h^2\right); A = 2\pi rh = 2\pi r^2\left(1 - \cos\theta\right) = \pi\left(a^2 + h^2\right); r = \frac{a^2 + h^2}{2h}$