Sogang University

Kim N Kang

Sangkeun Kim, Woohyun Kim, Mingyun Kang

Table of Contents

Graph Algorithm

Dijkstra’s Shortest Path

```
#include <queue>
#include <vector>
#include <algorithm>
using namespace std;
typedef pair<int,int> ii;
vector<vector<ii> > v;
vector<int> d;
const int inf=0x7FFFFFFF;
/*
 * v.resize(V+!),d.resize(V+1);
 */
int dijkstra(int s,int e) {
    priority_queue<ii,vector<ii>,greater<ii> > pq;
    fill(d.begin(),d.end(),inf);
    d[s]=0;
    pq.push(ii(d[s],s));
    while ( !pq.empty() ){
        ii now=pq.top();pq.pop();
        int cur=now.second;
        if ( d[cur] < now.first ) continue;
        for ( int i = 0 ; i < v[cur].size() ; i++ ) {
            ii next=v[cur][i];
            if ( d[next.first] > d[cur]+next.second ) {
                d[next.first] = d[cur]+next.second;
                pq.push(ii(d[next.first],next.first));
            }
        }
    }
    return d[e];
}
```

Strongly Connected Component & Bi-connected Component

```
cc::graph[x].push_back(y); // 정점 x와 y가 연결됨
result = cc::scc(size); // Strongly Connected Component의 개수
f = (connected[i] == connected[j]); // 정점 i와 j가 같은 SCC에 속하는가?
cc::bcc(size);
n = cc::cut_vertex_num; // 절점의 개수
b = cc::cut_vertex[i]; // 정점 i가 절점인가?
n = cc::cut_edge_num; // 절선의 개수
p = cc::cut_edge[i][0], q = cc::cut_edge[i][1]; // i번째 절선 p-q
```

```
#include <cstdlib>
#include <vector>
using namespace std;
namespace cc
{
    const int SIZE = 10000;
    vector<int> graph[SIZE];
    int connected[SIZE];
    int cut_vertex_num;
    bool cut_vertex[SIZE];
    int cut_edge_num, cut_edge[SIZE][2];
    int order[SIZE];
    int visit_time[SIZE], finish[SIZE], back[SIZE];
    int stack[SIZE], seen[SIZE];
#define MIN(a,b) (a) = ((a)<(b))?(a):(b)
    int dfs(int size) {
        int top, cnt, cnt2, cnt3;
        int i;
        cnt = cnt2 = cnt3 = 0;
        stack[0] = 0;
        for (i = 0 ; i < size ; i++) visit_time[i] = -1;
        for (i = 0 ; i < size ; i++) cut_vertex[i] = false; // CUT VERTEX
        cut_edge_num = 0; // CUT_EDGE
        for (i = 0 ; i < size ; i++) {
            if (visit_time[order[i]] == -1) {
                top = 1;
                stack[top] = order[i];
                seen[top] = 0;
                visit_time[order[i]] = cnt++;
                connected[order[i]] = cnt3++;
                int root_child = 0; // CUT VERTEX
                while (top > 0) {
                    int j, now = stack[top];
                  if (seen[top] == 0) back[now] = visit_time[now]; // NOT FOR SCC
                    for (j = seen[top] ; j < graph[now].size() ; j++) {
                        int next = graph[now][j];
                        if (visit_time[next] == -1) {
                            if (top == 1) root_child++; // CUT VERTEX
                            seen[top] = j + 1;
                            stack[++top] = next;
                            seen[top] = 0;
                            visit_time[next] = cnt++;
                            connected[next] = connected[now];
                            break;
                        }
                      else if (top == 1 || next != stack[top - 1]) // NOT FOR SCC
                            MIN(back[now], visit_time[next]); // NOT FOR SCC
                    }
                    if (j == graph[now].size()) {
                        finish[cnt2++] = now; // NOT FOR BCC
                        top--;
                        if (top > 1) {
                            MIN(back[stack[top]], back[now]); // NOT FOR SCC
```

```
                    if (back[now] >= visit_time[stack[top]]) { // CUT VERTEX
                        cut_vertex[stack[top]] = true;
                        cut_vertex_num++;
                    }
                }
                // CUT EDGE
                if (top > 0 && visit_time[stack[top]] < back[now]) {
                    cut_edge[cut_edge_num][0] = stack[top];
                    cut_edge[cut_edge_num][1] = now;
                    cut_edge_num++;
                }
            }
        }
        if (root_child > 1) { // CUT VERTEX
            cut_vertex[order[i]] = true;
            cut_vertex_num++;
        }
    }
}
return cnt3; // number of connected component
}
#undef MIN
    vector<int> graph_rev[SIZE];
    void graph_reverse(int size) {
        for (int i = 0 ; i < size ; i++) graph_rev[i].clear();
        for (int i = 0 ; i < size ; i++)
            for (int j = 0 ; j < graph[i].size() ; j++)
                graph_rev[graph[i][j]].push_back(i);
        for (int i = 0 ; i < size ; i++) graph[i] = graph_rev[i];
    }
    int scc(int size) {
        int n;
        for (int i = 0 ; i < size ; i++) order[i] = i;
        dfs(size);
        graph_reverse(size);
        for (int i = 0 ; i < size ; i++) order[i] = finish[size - i - 1];
        n = dfs(size);
        graph_reverse(size);
        return n;
    }
    void bcc(int size) {
        for (int i = 0 ; i < size ; i++) order [ i ] = i;
        dfs(size);
        cut_vertex_num = 0;
        for (int i = 0 ; i < size ; i++)
            if (cut_vertex[i])
                cut_vertex_num++;
    }
} // namespace cc
```

Min-cost Max-flow using bellman-ford algorithm

```
mcmf::init(graph, size); // 그래프 초기화
result = mcmf::maximum_flow(source, sink); // 최대 매칭, 최소 비용 pair
```

```
#include <cstring>
#include <vector>
#include <algorithm>
using namespace std;
struct edge {
    int target;
    int capacity; // cap_t
    int cost; // cost_t
};
namespace mcmf
{
    typedef int cap_t; // capacity type
    typedef int cost_t; // cost type
    const int SIZE = 300;
    const cap_t CAP_INF = 0x7fFFffFF;
    const cost_t COST_INF = 0x7fFFffFF;
    int n;
    vector<pair<pair<int, edge>, int> > g;
    int p[SIZE];
    cost_t dist[SIZE];
    cap_t mincap[SIZE];
    int pth[SIZE];
    void init(const vector<edge> graph[], int size) {
        int i, j;
        n = size;
        memset(p, -1, sizeof(p));
        g.clear();
        for (i = 0 ; i < size ; i++) {
            for (j = 0 ; j < graph[i].size() ; j++) {
                int next = graph[i][j].target;
                edge tmp = graph[i][j];
                g.push_back(make_pair(make_pair(i, tmp), p[i]));
                p[i] = g.size() - 1;
                tmp.target = i;
                tmp.capacity = 0;
                tmp.cost = -tmp.cost;
                g.push_back(make_pair(make_pair(next, tmp), p[next]));
                p[next] = g.size() - 1;
            }
        }
    }
    int bellman(int s, int t) {
        int i, j;
        for (i = 0 ; i < n ; i++) {
            dist[i] = COST_INF;
            mincap[i] = 0;
```

```
        }
        dist[s] = 0;
        mincap[s] = CAP_INF;
        bool flg = false;
        for (i = 0 ; i < n ; i++) {
            flg = false;
            for (j = 0 ; j < g.size() ; j++) {
                int now, next;
                if (g[j].first.second.capacity == 0) continue;
                now = g[j].first.first;
                next = g[j].first.second.target;
                if (dist[now] == COST_INF) continue;
                if (dist[now] + g[j].first.second.cost < dist[next]) {
                    dist[next] = dist[now] + g[j].first.second.cost;
                    pth[next] = j;
                    mincap[next] = min(mincap[now], g[j].first.second.capacity);
                    flg = true;
                }
            }
            if (!flg) break;
        }
        if (flg) return -1;
        return dist[t] != COST_INF ? 1 : 0;
    }
    pair<cap_t, cost_t> maximum_flow(int source, int sink) {
        cap_t total_flow = 0;
        cost_t total_cost = 0;
        int state;
        while ((state = bellman(source,sink)) > 0) {
            cap_t f = mincap[sink];
            total_flow += f;
            total_cost += f * dist[sink];
            for (int i = sink ; i != source; i = g[pth[i]].first.first) {
                g[pth[i]].first.second.capacity -= f;
                g[pth[i] ^ 1].first.second.capacity += f;
            }
        }
        if (state == -1) while (true); // it's NP-Hard
        return make_pair(total_flow, total_cost);
    }
} // namespace mcmf
```

Min-cost Max-flow using dijkstra algorithm

```
mcmf::init(graph, size); // 그래프 초기화
result = mcmf::maximum_flow(source, sink); // 최대 매칭, 최소 비용 pair
```

```
#include <cstring>
#include <queue>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;
struct edge {
    int target;
    int capacity; // cap_t
    int cost; // cost_t
};
namespace mcmf
{
    typedef int cap_t; // capacity type
    typedef int cost_t; // cost type
    const int SIZE = 5000;
    const cap_t CAP_INF = 0x7fFFffFF;
    const cost_t COST_INF = 0x7fFFffFF;
    int n;
    vector<pair<edge, int> > g;
    int p[SIZE];
    cost_t dist[SIZE];
    cap_t mincap[SIZE];
    cost_t pi[SIZE];
    int pth[SIZE];
    int from[SIZE];
    bool v[SIZE];
    void init(const vector<edge> graph[], int size){
        int i, j;
        n = size;
        memset(p, -1, sizeof(p));
        g.clear();
        for (i = 0 ; i < size ; i++) {
            for (j = 0 ; j < graph[i].size() ; j++) {
                int next = graph[i][j].target;
                edge tmp = graph[i][j];
                g.push_back(make_pair(tmp, p[i]));
                p[i] = g.size() - 1;
                tmp.target = i;
                tmp.capacity = 0;
                tmp.cost = -tmp.cost;
                g.push_back(make_pair(tmp, p[next]));
                p[next] = g.size() - 1;
            }
        }
    }
    int dijkstra(int s, int t) {
        typedef pair<cost_t, int> pq_t;
        priority_queue<pq_t, vector<pq_t>, greater<pq_t> > pq;
        int i;
        for (i = 0 ; i < n ; i++) {
            dist[i] = COST_INF;
            mincap[i] = 0;
            v[i] = false;
```

```
        }
        dist[s] = 0;
        mincap[s] = CAP_INF;
        pq.push(make_pair(0, s));
        while (!pq.empty()) {
            int now = pq.top().second;
            pq.pop();
            if (v[now]) continue;
            v[now] = true;
            for (i = p[now] ; i != -1 ; i = g[i].second) {
                int next = g[i].first.target;
                if (v[next]) continue;
                if (g[i].first.capacity == 0) continue;
                cost_t pot = dist[now] + pi[now] - pi[next] + g[i].first.cost;
                if (dist[next] > pot) {
                    dist[next] = pot;
                    mincap[next] = min(mincap[now], g[i].first.capacity);
                    pth[next] = i;
                    from[next] = now;
                    pq.push(make_pair(dist[next], next));
                }
            }
        }
        for (i = 0 ; i < n ; i++) pi[i] += dist[i];
        return dist[t] != COST_INF;
    }
    pair<cap_t, cost_t> maximum_flow(int source, int sink) {
        memset(pi, 0, sizeof(pi));
        cap_t total_flow = 0;
        cost_t total_cost = 0;
        while (dijkstra(source, sink)) {
            cap_t f = mincap[sink];
            total_flow += f;
            for (int i = sink ; i != source ; i = from[i]) {
                g[pth[i]].first.capacity -= f;
                g[pth[i] ^ 1].first.capacity += f;
                total_cost += g[pth[i]].first.cost * f;
            }
        }
        return make_pair(total_flow, total_cost);
    }
} // namespace mcmf
```

Network Flow

```
netflow::n = XX; // 정점 개수
netflow::capacity[i][j] = XX; // 정점 i에서 j로의 용량
result = netflow::maximum_flow(source, sink);
f = netflow::flow[i][j]; // 정점 i에서 j로 흐르는 유량
```

```
#include <cstring>
#include <queue>
using namespace std;
namespace netflow
{
    typedef int val_t;
    const int SIZE = 1000;
    const val_t INF = 0x7fFFffFF;
    int n;
    val_t capacity[SIZE][SIZE];
    val_t total_flow;
    val_t flow[SIZE][SIZE];
    int back[SIZE];
    inline val_t res(int a, int b) {
        return capacity[a][b] - flow[a][b];
    }
    val_t push_flow(int source, int sink) {
        memset(back, -1, sizeof(back));
        queue<int> q;
        q.push(source);
        back[source] = source;
        while (!q.empty() && back[sink] == -1) {
            int now = q.front();
            q.pop();
            for (int i = 0 ; i < n ; i++) {
                if (res(now, i) > 0 && back[i] == -1) {
                    back[i] = now;
                    q.push(i);
                }
            }
        }
        if (back[sink] == -1) return 0;
        int now, bef;
        val_t f = INF;
        for (now = sink ; back[now] != -1 ; now = back[now])
            f = min(f, res(back[now], now));
        for (now = sink ; back[now] != -1 ; now = back[now]) {
            bef = back[now];
            flow[bef][now] += f;
            flow[now][bef] = -flow[bef][now];
        }
        total_flow += f;
        return f;
    }
    val_t maximum_flow(int source, int sink) {
        memset(flow, 0, sizeof(flow));
        total_flow = 0;
        while (push_flow(source, sink));
        return total_flow;
    }
} // namespace netflow
```

Network-flow using DINIC algorithm

```
#include <cstdio>
#include <vector>
#include <limits>
#include <iostream>
#include <queue>

#pragma warning(disable:4996)
using namespace std;

struct NetworkFlow
{
    typedef long long Weight;
    struct Edge {
        int to; unsigned next;
        Weight cap, flow;

        Edge(int to, Weight cap, unsigned next = ~0) : to(to), cap(cap), flow(0),
next(next) {}
        inline Weight res() const { return cap - flow; }
    };

    int V;
    Weight totalFlow;
    vector<Edge> edges;
    vector<unsigned> G;
    NetworkFlow(int V) : V(V), G(V, ~0), totalFlow(0) {}

    // DINIC Algorithm
    vector<int> d;
    vector<unsigned> p;

    void addEdge(int a, int b, Weight cab, Weight cba = 0) {
        edges.push_back( Edge(b, cab, G[a]) );
        G[a] = edges.size() - 1;
        edges.push_back( Edge(a, cba, G[b]) );
        G[b] = edges.size() - 1;
    }

    bool levelGraph(int S, int T) {
        queue<int> q; q.push(S);
        d = vector<int>(V, -1);
        d[S] = 0;
        while(!q.empty() && d[T] == -1) {
            int u = q.front(); q.pop();
            for(unsigned i = G[u]; i != ~0; i = edges[i].next) {
                Edge &e = edges[i];
                int v = e.to;
                if(e.res() > 0 && d[v] == -1) { d[v] = d[u] + 1; q.push(v); }
            }
        }
        return d[T] != -1;
    }

    int pushFlow(int u, int T, Weight amt) {
        if(!amt || u == T) return amt;
        for(unsigned &i = p[u]; i != ~0; i = edges[i].next) {
            Edge &e = edges[i], &rev = edges[i ^ 1];

            int v = e.to;
            if(e.res() > 0 && d[u] + 1 == d[v]) {
                Weight f = pushFlow(v, T, min(e.res(), amt));
                if(f > 0) {
                    e.flow += f, rev.flow -= f;
                    return f;
                }
            }
        }
        return 0;
    }

    Weight maxFlow(int S, int T) {
        totalFlow = 0;
        while( levelGraph(S, T) ) {
            p = G;
            while(Weight f = pushFlow(S, T, numeric_limits<Weight>::max()))
                totalFlow += f;
        }
        return totalFlow;
    }
};


int main() {
    int n, m;
    scanf("%d%d", &n, &m);

    NetworkFlow nf(n);

    for(int i=1; i<=m; ++i) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        if(a == b) continue;
        --a; --b;

        nf.addEdge(a, b, c); // uni-directional
        nf.addEdge(a, b, c, c); // bi-directional
    }

    printf("%lld\n", nf.maxFlow(0, n-1));
    return 0;
}
```

Bipartite Matching Using DFS Only

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>
using namespace std;
#define MAX_V 1000
vector<vector<int> > v;
int backMatch[MAX_V*2+5];
bool visited[MAX_V*2+5];
bool dfs(int now) {
    if ( visited[now] ) return false;
    visited[now] = true;
    for ( int i = 0 ; i < v[now].size() ; i++ ) {
        int next = v[now][i];
        if ( backMatch[next] == -1 || dfs(backMatch[next]) ) {
            backMatch[next] = now;
            return true;
        }
    }
    return false;
}
int BipartiteMatching() {
    memset(backMatch,-1,sizeof(backMatch));
    int matched =0;
    for ( int i = 0 ; i < v.size() ; i++ ) {
        memset(visited,false,sizeof(visited));
        if ( dfs(i) ) matched++;
    }
    return matched;
}
```

Bipartite Matching Using Hopcroft-Karp Algorithm

```
#include <cstdio>
#include <queue>
#include <vector>
#include <algorithm>
using namespace std;
#define MAX_V 1004
const int inf = 987654321;
int N,M;
int used[MAX_V],match[MAX_V],d[MAX_V];
vector<vector<int> > v;
queue<int> q;
void bfs() {
    for ( int i = 1 ; i <= N ; i++ )
        d[i] = inf;
    for ( int i = 1 ; i <= N ; i++ )
        if ( !used[i] ) d[i] =0,q.push(i);
    while ( !q.empty() ) {
        int now = q.front();q.pop();
        for ( int i = v[now].size() ; i-- ; ) {
            int next = v[now][i];
            if ( match[next] && d[match[next]] == inf )
                d[match[next]] = d[now]+1,q.push(match[next]);
        }
    }
}
bool dfs(int now) {
    for ( int i = v[now].size() ; i-- ; ) {
        int next = v[now][i];
        if ( !match[next] || d[match[next]] == d[now]+1 && dfs(match[next]) ) {
            used[now] = true, match[next] = now;
            return true;
        }
    }
    return false;
}
int matching() {
    int ret=0;
    while ( true ) {
        bfs();
        int flow=0;
        for ( int i = 1 ; i<= N ; i++ )
            if ( !used[i] && dfs(i) ) flow++;
        ret += flow;
        if ( !flow ) break;
    }
    return ret;
}
```

Hungarian Method

```
hungarian::n = XX; // 정점 개수
hungarian::cost[i][j] = XX; // 비용 테이블
result = hungarian::hungarian(); // 최대 매칭
y = hungarian::xy[x]; // 정점 x와 연결된 정점 번호
x = hungarian::yx[y]; // 정점 y와 연결된 정점 번호
```

```
#include <cstring>
#include <queue>
#include <algorithm>
#include <limits>
using namespace std;
namespace hungarian
{
```

```cpp
typedef double val_t;
const int SIZE = 100;
const val_t INF = numeric_limits<double>::infinity();
// 두 값이 같은지 비교
inline bool eq(val_t a, val_t b) {
    static const double eps = 1e-9;
    return (a - eps < b && b < a + eps);
}
int n;
val_t cost[SIZE][SIZE];
int xy[SIZE], yx[SIZE];

int match_num;
val_t lx[SIZE], ly[SIZE];
bool s[SIZE], t[SIZE];
int prev[SIZE];

val_t hungarian() {
    memset(xy, -1, sizeof(xy));
    memset(yx, -1, sizeof(yx));
    memset(ly, 0, sizeof(ly));
    match_num = 0;
    int x, y;
    for (x = 0 ; x < n ; x++) {
        lx[x] = cost[x][0];
        for (y = 1 ; y < n ; y++)
            lx[x] = max(lx[x], cost[x][y]);
    }
    for (x = 0 ; x < n ; x++)
        for (y = 0 ; y < n ; y++)
            if (eq(cost[x][y], lx[x] + ly[y]) && yx[y] == -1) {
                xy[x] = y;
                yx[y] = x;
                match_num++;
                break;
            }
    while (match_num < n) {
        memset(s, false, sizeof(s));
        memset(t, false, sizeof(t));
        memset(prev, -1, sizeof(prev));
        queue<int> q;
        for (x = 0 ; x < n ; x++) {
            if (xy[x] == -1) {
                q.push(x);
                s[x] = true;
                break;
            }
        }
        bool flg = false;
        while (!q.empty() && !flg) {
            x = q.front();
            q.pop();
            for (y = 0 ; y < n ; y++) {
                if (eq(cost[x][y], lx[x] + ly[y])) {
                    t[y] = true;
                    if (yx[y] == -1) {
                        flg = true;
                        break;
                    }
                    if (!s[yx[y]]) {
                        s[yx[y]] = true;
                        q.push(yx[y]);
                        prev[yx[y]] = x;
                    }
                }
            }
        }
        if (flg) {
            int t1, t2;
            while (x != -1) {
                t1 = prev[x];
                t2 = xy[x];
                xy[x] = y;
                yx[y] = x;
                x = t1;
                y = t2;
            }
            match_num++;
        }
        else {
            val_t alpha = INF;
            for (x = 0 ; x < n ; x++) if (s[x])
                for (y = 0 ; y < n ; y++) if (!t[y])
                    alpha = min(alpha, lx[x] + ly[y] - cost[x][y]);
            for (x = 0 ; x < n ; x++) if (s[x]) lx[x] -= alpha;
            for (y = 0 ; y < n ; y++) if (t[y]) ly[y] += alpha;
        }
    }
    val_t ret = 0;
    for (x = 0 ; x < n ; x++)
        ret += cost[x][xy[x]];
    return ret;
}
}// namespace hungarian
```

Geometry

Convex Hull (Subset of Geometry Library)

```cpp
hull = convex_hull(points); // convex hull의 꼭지점 좌표 vector
```

정수 좌표를 사용하고 싶다면 모든 double을 int나 long long으로 치환하라.

```
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;
const double eps = 1e-9;
inline int diff(double lhs, double rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}
struct Point {
    double x, y;
    Point() {}
    Point(double x_, double y_): x(x_), y(y_) {}
};
inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(a.x * b.y + b.x * c.y + c.x * a.y
            - a.y * b.x - b.y * c.x - c.y * a.x, 0);
}
inline double dist2(const Point &a, const Point &b) {
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    return dx * dx + dy * dy;
}
struct PointSorter {
    Point origin;
    PointSorter(const vector<Point>& points) {
        origin = points[0];
        for (int i = 1 ; i < points.size() ; i++) {
            int det = diff(origin.x, points[i].x);
            if (det > 0)
                origin = points[i];
            else if (det == 0 && diff(origin.y, points[i].y) > 0)
                origin = points[i];
        }
    }
    bool operator()(const Point &a, const Point &b) {
        if (diff(b.x, origin.x) == 0 && diff(b.y, origin.y) == 0) return false;
        if (diff(a.x, origin.x) == 0 && diff(a.y, origin.y) == 0) return true;
        int det = ccw(origin, a, b);
        if (det == 0) return dist2(a, origin) < dist2(b, origin);
        return det < 0;
    }
};
vector<Point> convex_hull(vector<Point> points) {
    if (points.size() <= 3)
        return points;
    PointSorter cmp(points);
    sort(points.begin(), points.end(), cmp);
    vector<Point> ans;
    ans.push_back(points[0]);
    ans.push_back(points[1]);
    for(int i = 2 ; i < points.size() ; i++) {
        while (ans.size() > 1 &&
                ccw(ans[ans.size() - 2], ans[ans.size() - 1], points[i]) >= 0)
            ans.pop_back();
        ans.push_back(points[i]);
    }
    return ans;
}
```

General Geometry Library

```
#include <cmath>
#include <vector>
using namespace std;
const double eps = 1e-9;
inline int diff(double lhs, double rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}
inline bool is_between(double check, double a, double b) {
    if (a < b)
        return (a - eps < check && check < b + eps);
    else
        return (b - eps < check && check < a + eps);
}
struct Point {
    double x, y;
    Point() {}
    Point(double x_, double y_): x(x_), y(y_) {}
    bool operator==(const Point& rhs) const {
        return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
    }
    const Point operator+(const Point& rhs) const {
        return Point(x + rhs.x, y + rhs.y);
    }
    const Point operator-(const Point& rhs) const {
        return Point(x - rhs.x, y - rhs.y);
    }
    const Point operator*(double t) const {
        return Point(x * t, y * t);
    }
};
struct Circle {
    Point center;
    double r;
    Circle() {}
    Circle(const Point& center_, double r_): center(center_), r(r_) {}
};
struct Line {
```

```
    Point pos, dir;
    Line() {}
    Line(const Point& pos_, const Point& dir_): pos(pos_), dir(dir_) {}
};
inline double inner(const Point& a, const Point& b) {
    return a.x * b.x + a.y * b.y;
}
inline double outer(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}
inline int ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point - line.pos), 0);
}
inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b - a, c - a), 0);
}
inline double dist(const Point& a, const Point& b) {
    return sqrt(inner(a - b, a - b));
}
inline double dist2(const Point &a, const Point &b) {
    return inner(a - b, a - b);
}
inline double dist(const Line& line, const Point& point, bool segment = false) {
    double c1 = inner(point - line.pos, line.dir);
    if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
    double c2 = inner(line.dir, line.dir);
    if (segment && diff(c2, c1) <= 0) return dist(line.pos + line.dir, point);
    return dist(line.pos + line.dir * (c1 / c2), point);
}
bool get_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    ret = b.pos + b.dir * t2;
    return true;
}
bool get_segment_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t1 = -outer(b.pos - a.pos, b.dir) / mdet;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return false;
    ret = b.pos + b.dir * t2;
    return true;
}
const Point inner_center(const Point &a, const Point &b, const Point &c) {
    double wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
    double w = wa + wb + wc;
    return Point(
            (wa * a.x + wb * b.x + wc * c.x) / w,
            (wa * a.y + wb * b.y + wc * c.y) / w);
}
const Point outer_center(Point a, Point b, Point c) {
    b.x-=a.x;
    b.y-=a.y;
    c.x-=a.x;
    c.y-=a.y;

    return Point((c.y*(b.x*b.x+b.y*b.y)-b.y*(c.x*c.x+c.y*c.y))/(2*(b.x*c.y-
b.y*c.x))+a.x,(-c.x*(b.x*b.x+b.y*b.y)+b.x*(c.x*c.x+c.y*c.y))/(2*(b.x*c.y-
b.y*c.x))+a.y);
}
vector<Point> circle_line(const Circle& circle, const Line& line) {
    vector<Point> result;
    double a = 2 * inner(line.dir, line.dir);
    double b = 2 * (line.dir.x * (line.pos.x - circle.center.x)
            + line.dir.y * (line.pos.y - circle.center.y));
    double c = inner(line.pos - circle.center, line.pos - circle.center)
        - circle.r * circle.r;
    double det = b * b - 2 * a * c;
    int pred = diff(det, 0);
    if (pred == 0)
        result.push_back(line.pos + line.dir * (-b / a));
    else if (pred > 0) {
        det = sqrt(det);
        result.push_back(line.pos + line.dir * ((-b + det) / a));
        result.push_back(line.pos + line.dir * ((-b - det) / a));
    }
    return result;
}
vector<Point> circle_circle(const Circle& a, const Circle& b) {
    vector<Point> result;
    int pred = diff(dist(a.center, b.center), a.r + b.r);
    if (pred > 0) return result;
    if (pred == 0) {
        result.push_back((a.center * b.r + b.center * a.r) * (1 / (a.r + b.r)));
        return result;
    }
    double aa = a.center.x * a.center.x + a.center.y * a.center.y - a.r * a.r;
    double bb = b.center.x * b.center.x + b.center.y * b.center.y - b.r * b.r;
    double tmp = (bb - aa) / 2.0;
    Point cdiff = b.center - a.center;
    if (diff(cdiff.x, 0) == 0) {
        if (diff(cdiff.y, 0) == 0)
            return result; // if (diff(a.r, b.r) == 0): same circle
        return circle_line(a, Line(Point(0, tmp / cdiff.y), Point(1, 0)));
    }
    return circle_line(a,
            Line(Point(tmp / cdiff.x, 0), Point(-cdiff.y, cdiff.x)));
}
const Circle circle_from_3pts(const Point& a, const Point& b, const Point& c) {
    Point ba = b - a, cb = c - b;
    Line p((a + b) * 0.5, Point(ba.y, -ba.x));
    Line q((b + c) * 0.5, Point(cb.y, -cb.x));
```

```
    Circle circle;
    if (!get_cross(p, q, circle.center))
        circle.r = -1;
    else
        circle.r = dist(circle.center, a);
    return circle;
}
const Circle circle_from_2pts_rad(const Point& a, const Point& b, double r) {
    double det = r * r / dist2(a, b) - 0.25;
    Circle circle;
    if (det < 0)
        circle.r = -1;
    else {
        double h = sqrt(det);
        // center is to the left of a->b
        circle.center = (a + b) * 0.5 + Point(a.y - b.y, b.x - a.x) * h;
        circle.r = r;
    }
    return circle;
}
```

Polygon Cut

```
// left side of a->b
vector<Point> cut_polygon(const vector<Point>& polygon, Line line) {
    if (!polygon.size()) return polygon;
    typedef vector<Point>::const_iterator piter;
    piter la, lan, fi, fip, i, j;
    la = lan = fi = fip = polygon.end();
    i = polygon.end() - 1;
    bool lastin = diff(ccw_line(line, polygon[polygon.size() - 1]), 0) > 0;
    for (j = polygon.begin() ; j != polygon.end() ; j++) {
        bool thisin = diff(ccw_line(line, *j), 0) > 0;
        if (lastin && !thisin) {
            la = i;
            lan = j;
        }
        if (!lastin && thisin) {
            fi = j;
            fip = i;
        }
        i = j;
        lastin = thisin;
    }
    if (fi == polygon.end()) {
        if (!lastin) return vector<Point>();
        return polygon;
    }
    vector<Point> result;
    for (i = fi ; i != lan ; i++) {
        if (i == polygon.end()) {
            i = polygon.begin();
            if (i == lan) break;
        }
        result.push_back(*i);
    }
    Point lc, fc;
    get_cross(Line(*la, *lan - *la), line, lc);
    get_cross(Line(*fip, *fi - *fip), line, fc);
    result.push_back(lc);
    if (diff(dist2(lc, fc), 0) != 0) result.push_back(fc);
    return result;
}
```

Line Segment

```
struct Point{
    double x, y;
    struct Point operator+(struct Point A) {
        return {A.x + x, A.y + y};
    }
    struct Point operator-(struct Point A) {
        return {x - A.x, y - A.y};
    }
    struct Point operator*(double A) {
        return {x*A, y*A};
    }
    bool operator!=(struct Point A) {
        return (x != A.x || y != A.y);
    }
};
struct Segment{
    struct Point P0, P1;
};

#define SMALL_NUM   0.00000001 // anything that avoids division overflow
// dot product (3D) which allows vector operations in arguments
#define dot(u,v)   ((u).x * (v).x + (u).y * (v).y)
#define perp(u,v)  ((u).x * (v).y - (u).y * (v).x)  // perp product  (2D)

// inSegment(): determine if a point is inside a segment
//    Input:  a point P, and a collinear segment S
//    Return: 1 = P is inside S
//            0 = P is  not inside S
int inSegment( Point P, Segment S) {
    if (S.P0.x != S.P1.x) {    // S is not  vertical
        if (S.P0.x <= P.x && P.x <= S.P1.x)
            return 1;
        if (S.P0.x >= P.x && P.x >= S.P1.x)
            return 1;
    }
    else {    // S is vertical, so test y  coordinate
        if (S.P0.y <= P.y && P.y <= S.P1.y)
```

```
            return 1;
        if (S.P0.y >= P.y && P.y >= S.P1.y)
            return 1;
    }
    return 0;
}
//===================================================================

// intersect2D_2Segments(): find the 2D intersection of 2 finite segments
//    Input:  two finite segments S1 and S2
//    Output: *I0 = intersect point (when it exists)
//            *I1 =  endpoint of intersect segment [I0,I1] (when it exists)
//    Return: 0=disjoint (no intersect)
//            1=intersect  in unique point I0
//            2=overlap  in segment from I0 to I1
int intersect2D_2Segments( Segment S1, Segment S2, Point* I0, Point* I1 ) {
    Point    u = S1.P1 - S1.P0;
    Point    v = S2.P1 - S2.P0;
    Point    w = S1.P0 - S2.P0;
    double   D = perp(u,v);

    // test if  they are parallel (includes either being a point)
    if (abs(D) < SMALL_NUM) {           // S1 and S2 are parallel
        if (perp(u,w) != 0 || perp(v,w) != 0)  {
            return 0;                    // they are NOT collinear
        }
        // they are collinear or degenerate
        // check if they are degenerate  points
        double du = dot(u,u);
        double dv = dot(v,v);
        if (du==0 && dv==0) {            // both segments are points
            if (S1.P0 !=  S2.P0)         // they are distinct  points
                 return 0;
            *I0 = S1.P0;                 // they are the same point
            return 1;
        }
        if (du==0) {                     // S1 is a single point
            if  (inSegment(S1.P0, S2) == 0)  // but is not in S2
                 return 0;
            *I0 = S1.P0;
            return 1;
        }
        if (dv==0) {                     // S2 a single point
            if  (inSegment(S2.P0, S1) == 0)  // but is not in S1
                 return 0;
            *I0 = S2.P0;
            return 1;
        }
        // they are collinear segments - get  overlap (or not)
        double t0, t1;                    // endpoints of S1 in eqn for S2
        Point w2 = S1.P1 - S2.P0;
        if (v.x != 0) {
                 t0 = w.x / v.x;
                 t1 = w2.x / v.x;
        } else {
                 t0 = w.y / v.y;
                 t1 = w2.y / v.y;
        }
        if (t0 > t1) {                   // must have t0 smaller than t1
                 double t=t0; t0=t1; t1=t;    // swap if not
        }
        if (t0 > 1 || t1 < 0) {
            return 0;      // NO overlap
        }
        t0 = t0<0? 0 : t0;               // clip to min 0
        t1 = t1>1? 1 : t1;               // clip to max 1
        if (t0 == t1) {                  // intersect is a point
            *I0 = S2.P0 + v * t0;
            return 1;
        }

        // they overlap in a valid subsegment
        *I0 = S2.P0 + v * t0;
        *I1 = S2.P0 + v * t1;
        return 2;
    }

    // the segments are skew and may intersect in a point
    // get the intersect parameter for S1
    double     sI = perp(v,w) / D;
    if (sI < 0 || sI > 1)                // no intersect with S1
        return 0;

    // get the intersect parameter for S2
    double     tI = perp(u,w) / D;
    if (tI < 0 || tI > 1)                // no intersect with S2
        return 0;

    *I0 = S1.P0 + u * sI;                // compute S1 intersect point
    return 1;
}
```

Distance from a point to a line

```
#include <cmath>
#define SQ(x) ((x)*(x))
#define dist(a, b, c, d) sqrt(SQ((a)-(c)) + SQ((b)-(d)))
// find minimum distance between a line segment(x1, y1, x2, y2) and a point (px, py)
double segdist(double x1, double y1, double x2, double y2, double px, double py)
{
```

```
    double l2 = SQ(x1-x2) + SQ(y1-y2);
    if(l2 == 0.0) return dist(x1,y1,px,py);
    double t = ((px-x2) * (x1-x2) + (py-y2) * (y1-y2)) / l2;
    if(t < 0) return dist(x2,y2,px,py);
    if(t > 1) return dist(x1,y1,px,py);
    return dist(x2 + t*(x1-x2), y2 + t*(y1-y2), px, py);
}
```

Mathematical Stuffs

```
#include <cmath>
#include <climits>
#include <vector>
#include <algorithm>
using namespace std;
```

Modular Power

n^k mod m을 구한다.

```
long long power(long long n, long long k, long long m = LLONG_MAX) {
    long long ret = 1;
    while (k) {
        if (k & 1) ret = (ret * n) % m;
        n = (n * n) % m;
        k >>= 1;
    }
    return ret;
}
```

Great Common Divisor

a와 b의 최대공약수를 구한다.

Dependencies: -

```
long long gcd(long long a, long long b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

Extended GCD

ac + bd = gcd(a, b)가 되는 (c, d)를 찾는다.

Dependencies: -

```
pair<long long, long long> extended_gcd(long long a, long long b) {
    if (b == 0) return make_pair(1, 0);
    pair<long long, long long> t = extended_gcd(b, a % b);
    return make_pair(t.second, t.first - t.second * (a / b));
}
```

Modular Inverse

ax = gcd(a, m) (mod m)가 되는 x를 찾는다.

Dependencies: extended_gcd(a, b)

```
    long long modinverse(long long a, long long m) {
        return (extended_gcd(a, m).first % m + m) % m;
    }
```

Chinese Remainder Theorem

x = a (mod n)가 되는 x를 찾는다.

Dependencies: gcd(a, b), modinverse(a, m)

```
    long long chinese_remainder(long long *a, long long *n, int size) {
        if (size == 1) return *a;
        long long tmp = modinverse(n[0], n[1]);
        long long tmp2 = (tmp * (a[1] - a[0]) % n[1] + n[1]) % n[1];
        long long ora = a[1];
        long long tgcd = gcd(n[0], n[1]);
        a[1] = a[0] + n[0] / tgcd * tmp2;
        n[1] *= n[0] / tgcd;
        long long ret = chinese_remainder(a + 1, n + 1, size - 1);
        n[1] /= n[0] / tgcd;
        a[1] = ora;
        return ret;
    }
```

Binomial Calculation

nCm의 값을 구한다.

Dependencies: -

파스칼의 삼각형을 이용하거나, 미리 계산된 값을 가져오도록 이 함수를 수정하면 lucas_theorem, catalan_number 함수의 성능을 향상시킬 수 있다.

```
long long binomial(int n, int m) {
    if (n < m || n < 0) return 0;
    long long ans = 1, ans2 = 1;
    for (int i = 0 ; i < m ; i++) {
        ans *= n - i;
        ans2 *= i + 1;
    }
    return ans / ans2;
}
```

Lucas Theorem

nCm mod p의 값을 구한다.

Dependencies: binomial(n, m)

n, m은 문자열로 주어지는 정수이다. p는 소수여야 한다.

```
int lucas_theorem(const char *n, const char *m, int p) {
    vector<int> np, mp;
    int i;
    for (i = 0 ; n[i] ; i++) {
        if (n[i] == '0' && np.empty()) continue;
        np.push_back(n[i] - '0');
    }
    for (i = 0 ; m[i] ; i++) {
        if (m[i] == '0' && mp.empty()) continue;
        mp.push_back(m[i] - '0');
    }
    int ret = 1;
    int ni = 0, mi = 0;
    while (ni < np.size() || mi < mp.size()) {
        int nmod = 0, mmod = 0;
        for (i = ni ; i < np.size() ; i++) {
            if (i + 1 < np.size())
                np[i + 1] += (np[i] % p) * 10;
            else
                nmod = np[i] % p;
            np[i] /= p;
        }
        for (i = mi ; i < mp.size() ; i++) {
            if (i + 1 < mp.size())
                mp[i + 1] += (mp[i] % p) * 10;
            else
                mmod = mp[i] % p;
            mp[i] /= p;
        }
        while (ni < np.size() && np[ni] == 0) ni++;
        while (mi < mp.size() && mp[mi] == 0) mi++;
        ret = (ret * binomial(nmod, mmod)) % p;
    }
    return ret;
}
```

Catalan Number

Dependencies: binomial(n, m)

```
    long long catalan_number(int n) {
        return binomial(n * 2, n) / (n + 1);
    }
typedef long long ll;
#define mod 1000000007ll
ll factorial[2222222];
ll pow(ll a,int b) {
    if ( b == 0 ) return 1;
    if ( b == 1 ) return a%mod;
    ll t = pow(a,b/2);
    t = (t*t)%mod;
    return (b&1)?(t*a)%mod:t;
}
ll catalanNumber(int n) {
    return ((((factorial[2*n]*pow(factorial[n],mod-
2))%mod)*pow(factorial[n+1],mod-2))%mod)%mod;
}
int main() {
    factorial[0] = factorial[1] = 1;
    for ( int i = 2 ; i <= 2222222 ; i++ )
        factorial[i] = (factorial[i-1]*i)%mod;
}
```

Euler's Totient Function

phi(n), n 이하의 양수 중 n과 서로 소인 것의 개수를 구한다.

Dependencies: -

```
// phi(n) = (p_1 - 1) * p_1 ^ (k_1 - 1) * (p_2 - 1) * p_2 ^ (k_2-1)
long long euler_totient2(long long n, long long ps) {
    for (long long i = ps ; i * i <= n ; i++) {
        if (n % i == 0) {
            long long p = 1;
            while (n % i == 0) {
                n /= i;
                p *= i;
            }
            return (p - p / i) * euler_totient2(n, i + 1);
        }
        if (i > 2) i++;
    }
    return n - 1;
}
long long euler_totient(long long n) {
    return euler_totient2(n, 2);
}
```

Matrix Inverse

Dependencies: -

```
inline bool eq(double a, double b) {
    static const double eps = 1e-9;
    return fabs(a - b) < eps;
}
// returns empty vector if fails
vector<vector<double> > mat_inverse(vector<vector<double> > matrix, int n) {
    int i, j, k;
    vector<vector<double> > ret;
    ret.resize(n);
    for (i = 0 ; i < n ; i++) {
        ret[i].resize(n);
```

```
        for (j = 0 ; j < n ; j++)
            ret[i][j] = 0;
        ret[i][i] = 1;
    }
    for (i = 0 ; i < n ; i++) {
        if (eq(matrix[i][i],0)) {
            for (j = i + 1 ; j < n ; j++) {
                if (!eq(matrix[j][i], 0)) {
                    for (k = 0 ; k < n ; k++) {
                        matrix[i][k] += matrix[j][k];
                        ret[i][k] += ret[j][k];
                    }
                    break;
                }
            }
            if (j == n) {
                ret.clear();
                return ret;}
        }
        double tmp = matrix[i][i];
        for (k = 0 ; k < n ; k++) {
            matrix[i][k] /= tmp;
            ret[i][k] /= tmp;
        }
        for (j = 0 ; j < n ; j++) {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0 ; k < n ; k++) {
                matrix[j][k] -= matrix[i][k] * tmp;
                ret[j][k] -= ret[i][k] * tmp;
            }
        }
    }
    return ret;
}
```

Modular Matrix Inverse

```
Dependencies: modinverse(a, m)
    // returns empty vector if fails
    vector<vector<long long> > mat_inverse(vector<vector<long long> > matrix, int
n, long long mod) {
        int i, j, k;
        vector<vector<long long> > ret;
        ret.resize(n);
        for (i = 0 ; i < n ; i++) {
            ret[i].resize(n);
            for (j = 0 ; j < n ; j++)
                ret[i][j] = 0;
            ret[i][i] = 1 % mod;
        }
        for (i = 0 ; i < n ; i++) {
            if (matrix[i][i] == 0) {
                for (j = i + 1 ; j < n ; j++) {
                    if (matrix[j][i] != 0) {
                        for (k = 0 ; k < n ; k++) {
                            matrix[i][k] = (matrix[i][k] + matrix[j][k]) % mod;
                            ret[i][k] = (ret[i][k] + ret[j][k]) % mod;
                        }
                        break;
                    }
                }
                if (j == n) {
                    ret.clear();
                    return ret;
                }
            }
            long long tmp = modinverse(matrix[i][i], mod);
            for (k = 0 ; k < n ; k++) {
                matrix[i][k] = (matrix[i][k] * tmp) % mod;
                ret[i][k] = (ret[i][k] * tmp) % mod;
            }
            for (j = 0 ; j < n ; j++) {
                if (j == i) continue;
                tmp = matrix[j][i];
                for (k = 0 ; k < n ; k++) {
                    matrix[j][k] -= matrix[i][k] * tmp;
                    matrix[j][k] = (matrix[j][k] % mod + mod) % mod;
                    ret[j][k] -= ret[i][k] * tmp;
                    ret[j][k] = (ret[j][k] % mod + mod) % mod;
                }
            }
        }
        return ret;
    }
```

Matrix Determinants

```
Dependencies: -
double mat_det(vector<vector<double> > matrix, int n) {
    int i, j, k;
    double ret = 1;
    for (i = 0 ; i < n ; i++) {
        if (eq(matrix[i][i], 0)) {
            for (j = i + 1 ; j < n ; j++) {
                if (!eq(matrix[j][i], 0)) {
                    for (k = 0 ; k < n ; k++)
                        matrix[i][k] += matrix[j][k];
                    break;
                }
            }
            if (j == n)
                return 0;
        }
```

```
        double tmp = matrix[i][i];
        for (k = 0 ; k < n ; k++)
            matrix[i][k] /= tmp;
        ret *= tmp;
        for (j = 0 ; j < n ; j++) {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0 ; k < n ; k++)
                matrix[j][k] -= matrix[i][k] * tmp;
        }
    }
    return ret;
}
```

Kirchhoff's Theorem

주어진 그래프에서 가능한 신장트리의 경우의 수를 구한다.

```
Dependencies: mat_det(matrix, n)
    long long count_spantree(vector<int> graph[], int size) {
        int i, j;
        vector<vector<double> > matrix(size - 1);
        for (i = 0 ; i < size - 1 ; i++) {
            matrix[i].resize(size - 1);
            for (j = 0 ; j < size - 1 ; j++)
                matrix[i][j] = 0;
            for (j = 0 ; j < graph[i].size() ; j++) {
                if (graph[i][j] < size - 1) {
                    matrix[i][graph[i][j]]--;
                    matrix[i][i]++;
                }
            }
        }
        return (long long)(mat_det(matrix, size - 1) + 0.5);
    }
```

Gaussian Elimination

gaussian::run(size_eq, size_var, A, B, C);

A는 1차원 배열의 꼴로 주어지는 2차원 행렬이다. 배열 C의 값을 채워 넣는 루틴은 별도로 구현하라.

val_t로 double을 사용할 경우 abs 함수의 구현을 적절히 수정하라.

```
#include <algorithm>
using namespace std;
long long gcd(long long a, long long b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}
struct rational {
    long long p, q;
    void red() {
        if (q < 0) {
            p *= -1;
            q *= -1;
        }
        long long t = gcd((p >= 0 ? p : -p), q);
        p /= t;
        q /= t;
    }
    rational() {}
    rational(long long p_): p(p_), q(1) {}
    rational(long long p_, long long q_): p(p_), q(q_) { red(); }
    bool operator==(const rational& rhs) const {
        return p == rhs.p && q == rhs.q;
    }
    bool operator!=(const rational& rhs) const {
        return p != rhs.p || q != rhs.q;
    }
    bool operator<(const rational& rhs) const {
        return p * rhs.q < rhs.p * q;
    }
    const rational operator+(const rational& rhs) const {
        return rational(p * rhs.q + q * rhs.p, q * rhs.q);
    }
    const rational operator-(const rational& rhs) const {
        return rational(p * rhs.q - q * rhs.p, q * rhs.q);
    }
    const rational operator*(const rational& rhs) const {
        return rational(p * rhs.p, q * rhs.q);
    }
    const rational operator/(const rational& rhs) const {
        return rational(p * rhs.q, q * rhs.p);
    }
};
namespace gaussian
{
    typedef rational val_t;
    const val_t abs(const val_t& x) {
        return (x.p >= 0) ? x : rational(-x.p, x.q);
    }
#define GET(i, j, n) A[i * n + j]
    // return true when solution exists, false o/w.
    bool run(int size_eq, int size_var, val_t* A, val_t* B, val_t* C) {
        int i = 0, j = 0, k, l;
        int maxi;
        val_t temp_r;
        val_t* x;
        val_t* y;
        while (i < size_eq && j < size_var) {
            maxi = i;
            for (k = i + 1 ; k < size_eq ; k++)
                if (abs(GET(maxi, j, size_var)) < abs(GET(k, j, size_var)))
                    maxi = k;
```

```
            if (GET(maxi, j, size_var) != val_t(0)) {
                x = A + i * size_var;
                y = A + maxi * size_var;
                for (k = 0 ; k < size_var ; k++)
                    swap(*(x + k), *(y + k));
                swap(B[i], B[maxi]);
                temp_r = *(x + j);
                for (k = j ; k < size_var ; k++)
                    *(x + k) = *(x + k) / temp_r;
                B[i] = B[i] / temp_r;
                for (k = 0 ; k < size_eq ; k++) {
                    if (k == i) continue;
                    temp_r = GET(k, j, size_var);
                    for (l = j ; l < size_var ; l++)
                        GET(k, l, size_var) = GET(k, l, size_var)
                            - temp_r * GET(i, l, size_var);
                    B[k] = B[k] - GET(k, j, size_var) * B[i];
                }
                i++;
            }
            j++;
        }
        if (i < size_eq)
            for ( ; i < size_eq ; i++)
                if (B[i] != val_t(0)) return false;
        // C[...] := Case by case
        return true;
    }
#undef GET
} // namespace gaussian
```

Simplex Algorithm

```
n := number of constraints
m := number of variables
matrix[0] := maximize할 식의 계수
matrix[1~n] := constraints
solution := results
solution[n] := 원하는 식의 최대값
부등식의 우변(변수 없는 쪽)이 음이 아닌 수가 되도록 정리하여 대입한다.
ex) Maximize p = -2x + 3y
Constraints:     x + 3y ≤ 40
                2x + 4y ≥ 10
                x ≥ 0, y ≥ 0
n = 2, m = 2, matrix =   [ 2 -3 1 0 0 ] , c = [ 0 ]
                         [ 1 3 0 1 0 ]        [ 40]
                         [ 2 4 0 0 -1 ]       [ 10]
namespace simplex
{
    const int MAX_N = 50;
    const int MAX_M = 50;
    const double eps = 1e-9;
    inline int diff(double a, double b) {
        if (a - eps < b && b < a + eps) return 0;
        return (a < b) ? -1 : 1;
    }
    int n, m;
    double matrix[MAX_N + 1][MAX_M + MAX_N + 1];
    double c[MAX_N + 1];
    double solution[MAX_M + MAX_N + 1];
    int simplex() { // 0: found solution, 1: no feasible solution, 2: unbounded
        int i, j;
        while (true) {
            int nonfeasible = -1;
            for (j = 0 ; j <= n + m ; j++) {
                int cnt = 0, pos = -1;
                for (i = 0 ; i <= n ; i++) {
                    if (diff(matrix[i][j], 0)) {
                        cnt++;
                        pos = i;
                    }
                }
                if (cnt != 1)
                    solution[j] = 0;
                else {
                    solution[j] = c[pos] / matrix[pos][j];
                    if (solution[j] < 0) nonfeasible = i;
                }
            }
            int pivotcol = -1;
            if (nonfeasible != -1) {
                double maxv = 0;
                for (j = 0 ; j <= n+m ; j++) {
                    if (maxv < matrix[nonfeasible][j]) {
                        maxv = matrix[nonfeasible][j];
                        pivotcol = j;
                    }
                }
                if (pivotcol == -1) return 1;
            }
            else {
                double minv = 0;
                for (j = 0 ; j <= n + m ; j++) {
                    if (minv > matrix[0][j]) {
                        minv = matrix[0][j];
                        pivotcol = j;
                    }
                }
                if(pivotcol == -1) return 0;
            }
            double minv = -1;
            int pivotrow = -1;
```

```
            for (i = 0 ; i <= n ; i++) {
                if (diff(matrix[i][pivotcol], 0) > 0) {
                    double test = c[i] / matrix[i][pivotcol];
                    if (test < minv || minv < 0) {
                        minv = test;
                        pivotrow = i;
                    }
                }
            }
            if (pivotrow == -1) return 2;
            for (i = 0 ; i <= n ; i++) {
                if (i == pivotrow) continue;
                if (diff(matrix[i][pivotcol], 0)) {
                    double ratio = matrix[i][pivotcol] /
matrix[pivotrow][pivotcol];
                    for (j = 0 ; j <= n + m ; j++) {
                        if (j == pivotcol) {
                            matrix[i][j] = 0;
                            continue;
                        }
                        else
                            matrix[i][j] -= ratio * matrix[pivotrow][j];
                    }
                    c[i] -= ratio * c[pivotrow];
                }
            }
        }
    }
} // namespace simplex
```

Miscellaneous

Binary Indexed Tree

```
BIT::Init(size); // BIT initializing
BIT::Read(idx);  // Read
BIT::Update(idx,val); // Update

#include <vector>
using namespace std;
namespace BIT {
    typedef long long ll;
    int MAX;
    vector<ll> tree;
    void Init(int size) {
        MAX=size;
        tree.resize(MAX+1);
    }
    ll Read(int idx) {
        ll ret=0;
        while ( idx > 0 ) {
            ret += tree[idx];
            idx -= (idx & -idx);
        }
        return ret;
    }
    void Update(int idx,int val) {
        while ( idx < MAX ) {
            tree[idx] += val;
            idx += (idx & -idx);
        }
    }
}
```

Fenwick tree interval update

```
const int MAXN =  2222222;
int N;
int dataMul[MAXN*2],dataAdd[MAXN*2];

void internalUpdate(int at, int mul, int add) {
    while (at < MAXN) {
        dataMul[at] += mul;
        dataAdd[at] += add;
        at |= (at + 1);
    }
}
void update(int left, int right, int by) {
    internalUpdate(left, by, -by * (left - 1));
    internalUpdate(right, -by, by * right);
}
int query(int at) {
    int mul = 0;
    int add = 0;
    int start = at;
    while (at >= 0) {
        mul += dataMul[at];
        add += dataAdd[at];
        at = (at & (at + 1)) - 1;
    }
    return mul * start + add;
}
```

Union Find using disjoint-set

```
UnionFind::Init(size); // set initializing
UnionFind::Find(node); // find parent
UnionFind::MakeUnion(x,y); // union(x,y)

#include <vector>
```

```
#include <algorithm>
using namespace std;
namespace UnionFind{
    vector<int> rank;
    vector<int> u;
    void Init(int size) {
        rank.resize(size+1,0);
        u.resize(size+1,0);
        for ( int i = 0 ; i <= size ; i++ )
            u[i] = i;
    }
    int Find(int now) {
        return (u[now]==now)?now:(u[now]=Find(u[now]));
    }
    void MakeUnion(int x,int y) {
        x = Find(x); y = Find(y);
        if ( x == y ) return;
        if ( rank[x] < rank[y] ) u[x] = y;
        else {
            u[y] = x;
            rank[x]+=(rank[x]==rank[y]);
        }
    }
}
```

KMP Algorithm

```
result = kmp::match(text, pattern); // 모든 matched point의 vector

#include <vector>
using namespace std;
namespace kmp
{
    typedef vector<int> seq_t;
    void calculate_pi(vector<int>& pi, const seq_t& str) {
        pi[0] = -1;
        int j = -1;
        for (int i = 1 ; i < str.size() ; i++) {
            while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
            if (str[i] == str[j + 1])
                pi[i] = ++j;
            else
                pi[i] = -1;
        }
    }
    /* returns all positions matched */
    vector<int> match(seq_t text, seq_t pattern) {
        vector<int> pi(pattern.size());
        vector<int> ans;
        if (pattern.size() == 0) return ans;
        calculate_pi(pi, pattern);
        int j = -1;
        for (int i = 0 ; i < text.size() ; i++) {
            while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
            if (text[i] == pattern[j + 1]) {
                j++;
                if (j + 1 == pattern.size()) {
                    ans.push_back(i - j);
                    j = pi[j];
                }
            }
        }
        return ans;
    }
}
```

Suffix Array O(n log^2 n) with LCP

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
// L: doubling method 정렬을 위한 정보
// P[stp][i]: 길이가 1 << stp인 원래 문자열의 위치 i부터 시작하는 버켓 번호
int N, i, stp, cnt;
int A[65536];
struct entry {
    int nr[2], p;
} L[65536];
int P[17][65536];
int suffix_array[65536];
int lcp[65536]; // lcp(i, i + 1)
int cmp(struct entry a, struct entry b) {
    return (a.nr[0] == b.nr[0]) ? (a.nr[1] < b.nr[1]) : (a.nr[0] < b.nr[0]);
}
// calclcp(x, y) = min(lcp[x], lcp[x + 1], ..., lcp[y - 1])
// binary indexed tree needed for speedup
int calclcp(int x, int y) { // x, y: start position in original string
    int k, ret = 0;
    if(x == y) return N - x;
    for(k = stp - 1 ; k >= 0 && x < N && y < N ; k--)
        if(P[k][x] == P[k][y])
            x += 1 << k, y += 1 << k, ret += 1 << k;
    return ret;
}
int main(void) {
    int i;
    scanf("%d",&N);
    for(i = 0 ; i < N ; i++) {
        scanf("%d", &A[i]);
        P[0][i] = A[i];
```

```
    }
    for (stp = 1, cnt = 1 ; (cnt >> 1) < N ; stp++, cnt <<= 1) {
        for (i = 0 ; i < N ; i++) {
            L[i].nr[0] = P[stp - 1][i];
            L[i].nr[1] = (i + cnt < N) ? P[stp - 1][i + cnt] : -1;
            L[i].p = i;
        }
        sort(L, L + N, cmp);
        for (i = 0 ; i < N ; i++) {
            P[stp][L[i].p] = (i > 0 && L[i].nr[0] == L[i - 1].nr[0]
                    && L[i].nr[1] == L[i - 1].nr[1]) ? P[stp][L[i-1].p] : i;
        }
    }
    for (i = 0 ; i < N ; i++)
        suffix_array[P[stp - 1][i]] = i;
    for (i = 0 ; i + 1 < N ; i++)
        lcp[i] = calclcp(suffix_array[i], suffix_array[i + 1]);
    return 0;
}
```

Lowest Common Ancestor <O(n log n), O(log n)>

```
void Prepare_LCA(void)
{
    // pd : distance to parent, p : parent(direct), O(nlogn)
    memset(P, -1, sizeof P);
    for (int i = 1; i <= N; i++) {
        D[i][0] = pd[i];
        P[i][0] = p[i];
    }
    for (int j = 1; 1 << j <= N; j++) {
        for (int i = 1; i <= N; i++)
            if (P[i][j-1] != -1) {
                P[i][j] = P[P[i][j-1]][j-1];
                D[i][j] = D[P[i][j-1]][j-1] + D[i][j-1];
            }
    }
}

int Query_LCA(int x, int y)
{
    // O(logn)
    int log, ret = 0;
    if (lv[x] < lv[y]) swap(x, y);
    for (log = 1; 1 << log <= lv[x]; ++log); --log;
    for (int i = log; i >= 0; i--) {
        if (lv[x] - (1 << i) >= lv[y]) {
            ret += D[x][i];
            x = P[x][i];
        }
    }

    if (x == y) return ret;

    for (int i = log; i >= 0; i--) {
        if (P[x][i] != -1 && P[x][i] != P[y][i]) {
            ret += D[x][i] + D[y][i];
            x = P[x][i]; y = P[y][i];
        }
    } if (p[x] != p[y]) while (true); // NOT CONNECTED
    return ret + pd[x] + pd[y];
}
```

Pick's Theorem

On a simple polygon constructed on a grid of equal-distanced points, for area A, number of interior points I, number of boundary points B, we have A=I+B/2-1.

Combinatorial Game Theory

game sum: A xor B
game calc: minimum excluded number { Possible Games }
staircase nim: 짝수 계단에 있는 것들은 전부 소용 없음. 누구든 원래 nim 상태로 복귀시킬 수 있다.
Moore's nim_k: k개씩 제거하는 nim. 2진수로 변환하고, k+1진수에서 xor 하듯이 carry 없 이 더한다.
misere nim: play exactly as if you were playing normal play nim, except if your winning move would lead to a position that consists of heaps of size one only. In that case, leave exactly one more or one fewer heaps of size one than the normal play strategy recommends.

Combination Generator

```
/*
 * bit n개 중에 r개를 1로 바꿔준다.
 * while은 nCr만큼 돌고 x는 모든 경우의 수 비트를 갖는다.
 */
void combination_generator(int n,int r)
{
    int x, s, s1, t, k;

    x=(1<<r)-1;
    while(!(x & (1<<n))){
        s=x&-x;
```

```
        t=x+s;
        s1=t&-t;
        k=((s1/s)>>1)-1;
        x=t|k;
    }
}
```

Range MinMaximum Query Using Segment Tree

```
typedef pair<int,int> ii;
int a[1111111];
int mntree[4444444];
int mxtree[4444444];
void initialize(int node,int s,int e) {
    if ( s == e ) mntree[node] = mxtree[node] = a[s];
    else {
        int mid = (s+e)>>1;
        initialize(2*node,s,mid);
        initialize(2*node+1,mid+1,e);
        mxtree[node] = max(mxtree[2*node],mxtree[2*node+1]);
        mntree[node] = min(mntree[2*node],mntree[2*node+1]);
    }
}
ii query(int node,int s,int e,int i,int j) {
    if ( e < i || s > j ) return ii(-1,-1);
    if ( s >= i && e <= j ) return ii(mxtree[node],mntree[node]);
    int mid = (s+e)>>1;
    ii p1 = query(2*node,s,mid,i,j);
    ii p2 = query(2*node+1,mid+1,e,i,j);
    if ( p1 == ii(-1,-1) ) return p2;
    if ( p2 == ii(-1,-1) ) return p1;
    return
ii(max(max(0,p1.first),max(0,p2.first)),min(max(0,p1.second),max(0,p2.second)));
}
ii update(int node,int s,int e,int idx,int val) {
    if ( e < idx || idx < s ) return ii(mxtree[node],mntree[node]);
    if ( s == e ) return ii(mxtree[node]=val,mntree[node]=val);
    int mid = (s+e)>>1;
    ii p1 = update(2*node,s,mid,idx,val);
    ii p2 = update(2*node+1,mid+1,e,idx,val);
    return ii(mxtree[node]=max(max(0,p1.first),max(0,p2.first)),
            mntree[node]=min(max(0,p1.second),max(0,p2.second)));
}
```

Segment tree lazy propagation

```
const int MAXN =  1111111;
int N;
int tree[4*MAXN],lazy[4*MAXN];

void update(int left,int right,int node,int nodeLeft,int nodeRight,int val) {
    if ( nodeLeft > right || nodeRight < left ) return;
    if ( nodeLeft < nodeRight ) {
        lazy[2*node] += lazy[node];
        lazy[2*node+1] += lazy[node];
    }
    tree[node] += lazy[node];
    lazy[node] = 0;
    if ( left <= nodeLeft && nodeRight <= right ) {
        tree[node] += val;
        if ( nodeLeft < nodeRight ) {
            lazy[2*node] += val;
            lazy[2*node+1] += val;
        }
    } else if ( nodeLeft < nodeRight ) {
        int mid = (nodeLeft+nodeRight)>>1;
        update(left,right,node*2,nodeLeft,mid,val);
        update(left,right,node*2+1,mid+1,nodeRight,val);
        tree[node] = max(tree[node*2],tree[node*2+1]);
    }
}
int query(int left,int right,int node,int nodeLeft,int nodeRight) {
    if ( nodeLeft > right || nodeRight < left ) return 0;
    if ( nodeLeft < nodeRight ) {
        lazy[2*node] += lazy[node];
        lazy[2*node+1] += lazy[node];
    }
    tree[node] += lazy[node];
    lazy[node] = 0;
    int ret = 0;
    if ( left <= nodeLeft && nodeRight <= right ) return tree[node];
    else if ( nodeLeft < nodeRight ) {
        int mid = (nodeLeft+nodeRight)>>1;
        ret = max(ret, query(left,right,node*2,nodeLeft,mid));
        ret = max(ret, query(left,right,node*2+1,mid+1,nodeRight));
        tree[node] = max(tree[node*2],tree[node*2+1]);
    }
    return ret;
}
```

AntiPodal Point

컨벡스 헐로 구한 점들 중 가장 먼 두 점을 구한다. (C++11)

```
Dependencies : convex_hull
pair<Point,Point> AntiPodal(vector <Point>&& v)
{
    int n = v.size(), ans = 0;
    if (n < 3) return {v[0], v[1]};
    Point p1, p2;

    int p = n-1;
    int q = Next(p);
    while (abs(CCW(v[p], v[Next(p)], v[Next(q)])) > abs(CCW(v[p], v[Next(p)],
```

```
v[q]))) {
        q = Next(q);
    }

    int q0 = q;

    while (q != 0) {
        p = Next(p);
        if (ans < Dist(v[p], v[q])) {
            ans = Dist(v[p], v[q]);
            p1 = v[p], p2 = v[q];
        }// Found
        while (abs(CCW(v[p], v[Next(p)], v[Next(q)])) > abs(CCW(v[p], v[Next(p)],
v[q]))) {
            q = Next(q);
            if (v[p] != v[q0] || v[q] != v[0]) {
                if (ans < Dist(v[p], v[q])) {
                    ans = Dist(v[p], v[q]);
                    p1 = v[p], p2 = v[q];
                }// Found
            }
            else return {p1, p2};
        }

        if (abs(CCW(v[p], v[Next(p)], v[Next(q)])) == abs(CCW(v[p], v[Next(p)],
v[q]))) {
            if (v[p] != v[q0] || v[q] != v[n-1]) {
                if (ans < Dist(v[p], v[Next(q)])) {
                    ans = Dist(v[p], v[Next(q)]);
                    p1 = v[p], p2 = v[Next(q)];
                }// Found
            }
            else {
                if (ans < Dist(v[Next(p)], v[q])) {
                    ans = Dist(v[Next(p)], v[q]);
                    p1 = v[Next(p)], p2 = v[q];
                } // Found
            }
        }
    }

    return {p1, p2};
}
```

Aho-Corasick

```
#include <map>
#include <queue>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;
struct NODE {
    bool b;
    NODE *next[4], *f;
    NODE(){}
};
NODE *root;
NODE container[1111111];
int size;
NODE *newNode() {
    NODE *ret = &container[size++];
    ret->b = ret->f = 0;
    for ( int i = 0 ; i < 4; i++ )
        ret->next[i] = 0;
    return ret;
}
map<char,int> mp;
void createTree(vector<string>& pattern) {
    mp['A'] = 0;mp['C'] = 1;mp['G'] = 2;mp['T'] = 3;
    size = 0;
    root = newNode();
    for ( int i = 0 ; i < (int)pattern.size() ; i++ ) {
        NODE *now = root;
        for ( int j = 0 ; j < (int)pattern[i].length() ; j++ ) {
            int c = mp[pattern[i][j]];
            if ( !now->next[c] ) now->next[c] = newNode();
            now = now->next[c];
        }
        now->b = true;
    }

    queue<NODE*> q;
    for ( int i = 0 ; i < 4 ; i++ )
        if ( root->next[i] ) {
            root->next[i]->f = root;
            q.push(root->next[i]);
        }
    while ( !q.empty() ) {
        NODE *now = q.front();q.pop();
        NODE *f = now->f;
        for ( int i = 0 ; i < 4 ; i++ )
            if ( now->next[i] ) {
                NODE* &nf = now->next[i]->f;
                nf = f;
                while ( nf != root && !nf->next[i] )
                    nf = nf->f;
                if ( nf->next[i] ) nf = nf->next[i];
                q.push(now->next[i]);
            }
    }
}
vector<int> aho_corasick(string s) {
```

```
    vector<int> ret;
    NODE *now = root;
    int ans = 0;
    for ( int i = 0 ; i < (int)s.length() ; i++ ) {
        int c = mp[s[i]];
        while ( now != root && !now->next[c] ) now = now->f;
        if ( now->next[c] ) now = now->next[c];
        if ( now->b ) {
            ret.push_back(i);
            now = now->f;
        }
    }
    return ret;
}
```

Ternary Search

```
double f(double x) { return 0; }
// find maximum x
double ternary(double min, double max) {
    while(max - min > max * 1e-9) {
        double a = (min*2 + max) / 3.0;
        double b = (min + max*2) / 3.0;
        if(f(a) < f(b))
            min = a;
        else
            max = b;
    }
    return (min+max)*.5;
}
```

Hungraian Method

```
// Verified By UVA 11383 - Golden Tiger Claw
#include <string.h>
#include <algorithm>
using namespace std;
#define INF (1<<30)
#define MAX_N   1111
struct hungarian {
    //Inputs//////////////
    int N;
    int cost[MAX_N][MAX_N];
    ////////////////////

    int X[MAX_N], Y[MAX_N], Lx[MAX_N], Ly[MAX_N], Q[MAX_N], prev[MAX_N], res;
    int maxw_bipartite() {
        int tail, s, k;
        memset(Ly, 0, sizeof(int)*N);
        for(int i = 0 ; i < N ; i++) Lx[i] = *max_element(cost[i], cost[i] + N);
        memset(X, -1, sizeof(int)*N);
        memset(Y, -1, sizeof(int)*N);
        for(int i = 0; i < N; i++) {
            int head;
            memset(prev, -1, sizeof(int)*N);
            for (Q[0] = i, head = 0, tail = 1; head < tail && X[i] < 0; head++) {
                s = Q[head];
                for(int j = 0; j < N ; j++) {
                    if(X[i] >= 0) break;
                    if (Lx[s] + Ly[j] > cost[s][j] || prev[j] >= 0) continue;
                    Q[tail++] = Y[j];
                    prev[j] = s;
                    if (Y[j] < 0) while (j >= 0) {
                        s = prev[j]; Y[j] = s; k = X[s]; X[s] = j; j = k;
                    }
                }
            }
            if(X[i] < 0 && i-- + (k = INF)) {
                for(int head = 0 ; head < tail ; head++)
                    for(int j = 0 ; j < N ; j++)
                        if(prev[j] == -1) k = min(k, Lx[Q[head]] + Ly[j] -
cost[Q[head]][j]);
                for(int j = 0; j < tail ; j++) Lx[Q[j]] -= k;
                for(int j = 0; j < N ; j++) if (prev[j] >= 0) Ly[j] += k;
            }
        }
        res = 0;
        for(int i = 0 ; i < N ; i++) if(X[i] >= 0) res += cost[i][X[i]];
        return res;
    }
} w_match;
```

Network flow - Ford Fulkerson adj list

```
struct NetworkFlow {
  typedef int Weight;

  struct Edge {
    int a, b, next;    // a = from, b = to, next = next in adj list
    Weight c, f; // c = capacity, f = flow

    Edge() : a(-1), b(-1), c(0), f(0) {}
    Edge(int a, int b, Weight c, int next = -1) : a(a), b(b), c(c), f(0),
next(next) {}

    inline Weight res() const { return c - f; } // return residual cost
  };

  int V;
  Weight totalFlow, inf;
  vector<Edge> edges;
  vector<int> G; // G[v] = head edge from v

  //  Ford-Fulkerson Algorithm
```

```
  vector<bool> visited;

  NetworkFlow(int V) : V(V), G(V, -1), totalFlow(0) {
    inf = numeric_limits<Weight>::max();
  }

  void addEdge(int a,  int b, Weight cab, Weight cba = 0)  {
    edges.push_back( Edge(a, b, cab,  G[a]) );
    G[a] = (int)edges.size() - 1;
    edges.push_back( Edge(b, a, cba,  G[b]) );
    G[b] = (int)edges.size() - 1;
  }

  Weight dfs(int S,  int T, Weight amt)  {
    if (S == T) return amt;
    visited[S] = true;

    for (int i = G[S]; i != -1; i = edges[i].next)  {
      Edge &e = edges[i],  &rev = edges[i ^ 1];
      int v = e.b;
      if (e.res() > 0 && !visited[v]) {
        Weight flow = dfs(v, T, min(e.res(), amt));
        if (flow > 0) {
          e.f += flow, rev.f -= flow;
          return flow;
        }
      }
    }

    return 0;
  }

  Weight  maxFlow(int S, int T)  {
    totalFlow = 0;
    while( true ) {
      visited = vector<bool>(V, false);
      Weight flow = dfs(S, T, inf);
      if (flow == 0) break;
      totalFlow += flow;
    }

    return totalFlow;
  }
};
```