Seoul National University
rand().teamnote

Heon Young Yeom, Jin Yung Kim, Chan Min Kim, Gangwon Jo

Table of Context

Setting 2
    .vimrc 2
Quick STL Tutorial 2
    vector 2
    map 2
    set 2
    queue 3
    priority_queue 3
    string 3
    algorithm 3
Graph Algorithms 4
    Strongly Connected Component & Bi-connected Component 4
    Network Flow 5
    Network Flow Speedup 6
    Bipartite Matching 7
    Hungarian Method 8
    Min-cost Max-flow using Bellman-ford Algorithm 9
    Min-cost Max-flow using Dijkstra Algorithm 10
Mathematical Stuffs 11
    Modular Power 11
    Great Common Divisor 12
    Extended GCD 12
    Modular Inverse 12
    Chinese Remainder Theorem 12
    Binomial Calculation 12
    Lucas Theorem 12
    Catalan Number 13
    Euler’s Totient Function 13
    Matrix Inverse 13
    Modular Matrix Inverse 14
    Matrix Determinants 14
    Kirchhoff’s Theorem 15
    Gaussian Elimination 15
    Simplex Algorithm 16
Geometry 17
    Convex Hull (Subset of Geometry Library) 17
    General Geometry Library 18
    Polygon Cut 20
Miscellaneous 21
    Binary Indexed Tree 21
    Binary Indexed Tree Advanced 21
    KMP Algorithm 23
    Suffix Array O(n log n) 23
    Suffix Array O(n log^2 n) with LCP 24
    Pick’s Theorem 25
    Combinatorial Game Theory 25

Setting

.vimrc

```
syntax on
set ai si sw=4 ts=4
set nu ru
set backspace=start,indent,eol
set showmatch
set scrolloff=3
colorscheme evening
highlight linenr ctermbg=darkblue

set mouse=a
set hi=100
```

Quick STL Tutorial

vector

```
#include <vector>
#include <algorithm>
using namespace std;
#define ASSERT(x) if (!(x)) while (true);
int main() {
    vector<int> a; // empty vector
    vector<int> b(100); // b.size() == 100
    vector<int> c(100, 0); // c.size() == 100, c[i] == 0
    a.push_back(1);
    a.push_back(3);
    a.push_back(5); // a == {1, 3, 5}
    a.pop_back(); // a == {1, 3}
    for (int i = 0 ; i < a.size() ; i++)
        int x = a[i]; // do something
    ASSERT(find(a.begin(), a.end(), 3) != a.end());
    a.insert(find(a.begin(), a.end(), 1), 2); // a == {2, 1, 3}
    a.erase(find(a.begin(), a.end(), 3)); // a == {2, 1}
    if (!a.empty())
        a.clear();
    return 0;
}
```

map

```
#include <map>
#include <string>
using namespace std;
#define ASSERT(x) if (!(x)) while (true);
int main() {
    map<string, int> m;
    string k1 = "this is key";
    string k2 = "this is another key";
    m[k1] = 1;
    m[k2] = 2;
    m.erase(k2);
    ASSERT(m.find(k1) != m.end());
    for (map<string, int>::iterator it = m.begin() ; it != m.end() ; it++) {
        string k = it->first;
        int v = it->second; // do something
    }
    map<string, int>::iterator jt
        = m.lower_bound(string("this")); // jt->first >= "this"
    if (!m.empty())
        m.clear();
    return 0;
}
```

set

```
#include <set>
using namespace std;
#define ASSERT(x) if (!(x)) while (true);
int main() {
    set<int> s;
    s.insert(5);
    s.insert(3);
    s.insert(3); // nothing happened
    s.erase(3);
    s.erase(2); // it's ok.
    ASSERT(s.find(5) != s.end());
    for (set<int>::iterator it = s.begin() ; it != s.end() ; it++)
        int x = *it; // do something
    set<int>::iterator jt = s.lower_bound(4); // *jt >= 4
    if (!s.empty())
        s.clear();
    return 0;
}
```

queue

```
#include <queue>
using namespace std;
#define ASSERT(x) if (!(x)) while (true);
int main() {
    queue<int> q;
    q.push(5);
    q.push(3);
    ASSERT(q.front() == 5);
    q.pop();
    ASSERT(q.front() == 3);
    q.pop();
    ASSERT(q.empty());
    return 0;
}
```

priority_queue

```
#include <queue>
#include <vector>
#include <functional>
using namespace std;
#define ASSERT(x) if (!(x)) while (true);
struct my_struct {
    int e;
    bool operator<(const my_struct& rhs) const { // for max heap
        return e < rhs.e;
    }
    bool operator>(const my_struct& rhs) const { // for min heap
        return e > rhs.e;
    }
};
int main() {
    priority_queue<int> a; // max heap
    priority_queue<int, vector<int>, greater<int> > b; // min heap
    priority_queue<my_struct, vector<my_struct>, greater<my_struct> > c;
    a.push(5);
    a.push(3);
    ASSERT(a.top() == 5);
    a.pop();
    ASSERT(a.top() == 3);
    a.pop();
    ASSERT(a.empty());
    return 0;
}
```

string

```
#include <string>
using namespace std;
#define ASSERT(x) if (!(x)) while (true);
int main() {
    string a = "acm icpc world";
    ASSERT(a.size() == 14);
    ASSERT(!a.empty());
    a += " final"; // a == "acm icpc world final"
    a.push_back('.'); // a == "acm icpc world final."
    ASSERT(a.find("icpc") == 4);
    ASSERT(a.find("acm", 5) == string::npos);
    ASSERT(a.find_first_of("ijk") == 4);
    ASSERT(a.find_first_not_of("abcde") == 2);
    ASSERT(a.find_first_of(' ', 4) == 8);
    string b = a.substr(4, 4); // b == "icpc"
    string c = a.substr(15); // c == "final."
    a.insert(a.size() - 1, " 2010"); // a == "acm icpc world final 2010."
    a.erase(14); // a == "acm icpc world final"
    a.replace(9, a.size() - 9, "WF"); // a == "acm icpc WF"
    a.erase(3); // a == "acm"
    printf("%s\n", a.c_str());
    return 0;
}
```

algorithm

```
#include <algorithm>
#include <vector>
#include <functional>
using namespace std;
struct my_sorter {
    bool operator()(int a, int b) {
        return a < b;
    }
};
int main() {
    vector<int> a;
    a.push_back(1);
    a.push_back(5);
    a.push_back(3);
    a.push_back(4);
    a.push_back(2);
    sort(a.begin(), a.end());
    sort(a.begin(), a.end(), greater<int>());
    my_sorter cmp;
    sort(a.begin(), a.end(), cmp);
    unique(a.begin(), a.end());
    do {
        ; // do something
    } while (next_permutation(a.begin(), a.end()));
    return 0;
}
```

Graph Algorithms

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
```

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 Speedup

```
mcmf::init(graph, size); // 그래프 초기화
result = netflow::maximum_flow(source, sink);
f = netflow::flow[i][j]; // 정점 i에서 j로 흐르는 유량
```

```
#include <cstring>
#include <vector>
#include <queue>
using namespace std;

struct edge {
    int target;
    int capacity; // cap_t
};

namespace netflow
{
typedef int cap_t; // capacity type

const int SIZE = 5000;
const cap_t CAP_INF = 0x7fFFffFF;

int n;
vector<pair<edge, int> > g;
int p[SIZE];
int dist[SIZE];
cap_t maxcap;

void init(const vector<edge> graph[], int size) {
    int i, j;
    n = size;
    memset(p, -1, sizeof(p));
    maxcap = 0;
    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];
            maxcap = max(maxcap, tmp.capacity);
            g.push_back(make_pair(tmp, p[i]));
            p[i] = g.size() - 1;
            tmp.target = i;
            tmp.capacity = 0;
            g.push_back(make_pair(tmp, p[next]));
            p[next] = g.size() - 1;
        }
    }
}

bool bfs(int s,int t,int delta) {
    for (int i = 0 ; i < n ; i++)
        dist[i] = n + 1;
    queue<int> q;
    dist[s] = 0;
    q.push(s);
    while (!q.empty()) {
        int now = q.front();
        q.pop();
        for (int i = p[now] ; i != -1 ; i = g[i].second) {
            int next = g[i].first.target;
            if (g[i].first.capacity < delta) continue;
            if (dist[next] == n + 1) {
                dist[next] = dist[now] + 1;
                q.push(next);
            }
        }
    }
    return dist[t] != n + 1;
}
cap_t dfs(int now, int t, int delta, cap_t minv = CAP_INF) {
    if (now == t) return minv;
    for (int i = p[now] ; i != -1 ; i = g[i].second) {
        if (g[i].first.capacity < delta) continue;
        int next = g[i].first.target;
        if (dist[next] == dist[now] + 1) {
            cap_t flow = dfs(next, t, delta, min(minv, g[i].first.capacity));
            if (flow) {
                g[i].first.capacity -= flow;
                g[i ^ 1].first.capacity += flow;
                return flow;
            }
        }
    }
    return 0;
}

cap_t maxflow(int s, int t) {
    cap_t delta = 1, totalflow = 0;
    while (delta <= maxcap) delta <<= 1;
    while (delta >>= 1) {
        while (bfs(s, t, delta)) {
            cap_t flow;
            while (flow = dfs(s, t, delta)) // not ==
                totalflow += flow;
        }
    }
    return totalflow;
}

} // namespace netflow
```

Bipartite Matching

```
matching::v1 = XX; matching::v2 = XX; // 정점 개수
matching::graph[x].push_back(y); // 정점 x와 y가 연결됨
result = matching::hopcroft(); // 매칭 수
y = matching::mx[x]; // 정점 x와 연결된 정점 번호
x = matching::my[y]; // 정점 y와 연결된 정점 번호
```

```
#include <cstring>
#include <vector>
#include <queue>
using namespace std;

namespace matching
{
typedef int val_t;
const int SIZE = 1000;

int v1, v2;
vector<int> graph[SIZE];
int mx[SIZE], my[SIZE];
int total_matching;

int dist[SIZE];
int inf_dist;

bool bfs() {
    int x, y;
    queue<int> q;
    for (x = 0 ; x < v1 ; x++) {
        if (mx[x] == -1) {
            dist[x] = 0;
            q.push(x);
        }
        else
            dist[x] = -1;
    }
    bool flg = false;
    while (!q.empty()) {
        x = q.front();
        q.pop();
        for (int i = 0 ; i < graph[x].size() ; i++) {
            y = graph[x][i];
            if (my[y] == -1) {
                inf_dist = dist[x] + 1;
                flg = true;
            }
            else if (dist[my[y]] == -1) {
                dist[my[y]] = dist[x] + 1;
                q.push(my[y]);
            }
        }
    }
    return flg;
}

bool dfs(int x) {
    if (x == -1) return true;
    for (int i = 0 ; i < graph[x].size() ; i++) {
        int y = graph[x][i];
        int tmp = (my[y] == -1) ? inf_dist : dist[my[y]];
        if (tmp == dist[x] + 1 && dfs(my[y])) {
            mx[x] = y;
            my[y] = x;
            return true;
        }
    }
    dist[x] = -1;
    return false;
}

int hopcroft() {
    memset(mx, -1, sizeof(mx));
    memset(my, -1, sizeof(my));
    total_matching = 0;
    while (bfs()) {
        for (int x = 0 ; x < v1 ; x++)
            if (mx[x] == -1 && dfs(x))
                total_matching++;
    }
    return total_matching;
}

} // namespace matching
```

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
{
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
```

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
```

Mathematical Stuffs

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

Modular Power

n^k mod m을 구한다.

Dependencies: -

```
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);
}
```

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
```

Geometry

Convex Hull (Subset of Geometry Library)

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(const Point &a, const Point &b, const Point &c) {
    Point d1 = b - a, d2 = c - a;
    double area = outer(d1, d2);
    double dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y
        + d1.y * d2.y * (d1.y - d2.y);
    double dy = d1.y * d1.y * d2.x - d2.y * d2.y * d1.x
        + d1.x * d2.x * (d1.x - d2.y);
    return Point(a.x + dx / area / 2.0, a.y - dy / area / 2.0);
}

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;
}
```

Miscellaneous

Binary Indexed Tree

```
itree::init();
itree::update(pos, val); // pos 위치를 val로 업데이트
val = itree::getrange(s, e); // [s, e] 구간의 대표값
```

```
namespace itree
{
typedef int val_t;
const int size = 16384; // 2의 제곱수여야 함

// 트리를 초기화할 값
// 예) 구간 최소: 0x7fFFffFF
//     구간합: 0
const val_t init_value = 0;

// 트리의 두 child를 병합하는 함수
// 예) 구간 최소: return min(a, b);
//     구간합: return a + b;
val_t sum(val_t a, val_t b) {
    return a + b;
}

val_t itree[size * 2 + 1];

void init() {
    for (int i = 1 ; i <= size * 2 ; i++)
        itree[i] = init_value;
}

void update(int pos, val_t val) {
    pos |= size;
    itree[pos] = val;
    while (pos >>= 1)
        itree[pos] = sum(itree[pos << 1], itree[pos << 1 | 1]);
}

val_t getrange(int s, int e) { // [s, e]
    val_t ret = init_value;
    s |= size;
    e |= size;
    while(s <= e) {
        if(s & 1)
            ret = sum(ret, itree[s]);
        if((e & 1) == 0)
            ret = sum(ret, itree[e]);
        s = (s + 1) >> 1;
        e = (e - 1) >> 1;
    }
    return ret;
}

} // namespace itree
```

Binary Indexed Tree Advanced

```
itree::init();
itree::update(s, e, val); // [s, e] 구간의 값을 val로 업데이트
val = itree::getrange(s, e); // [s, e] 구간의 대표값
```

```
#include <algorithm>
using namespace std;

namespace itree
{
typedef int val_t;
const int size = 1024; // 2의 제곱수여야 함

// 트리를 초기화할 값
// 예) 구간 최소: 0x7fFFffFF
//     구간합: 0
const val_t init_value = 0;

// 내부노드 갱신을 위해 가중치를 계산하는 함수
// 예) 구간 최소/최대: return a;
//     구간 합: return a * len;
inline val_t weight(val_t a, int len) {
    return a * len;
```

```
}

// 트리의 두 child를 병합하는 함수
// 예) 구간 최소: return min(a, b);
//     구간합: return a + b;
val_t sum(val_t a, val_t b) {
    return a + b;
}

// 노드의 구간 대표값: 두 child를 병합한 값 a와, 자신에게 할당된 값 b를 병합
// 예) 구간 최소: return min(a, b);
//     구간합: return a + b;
val_t update_a(val_t a, val_t b) {
    return a + b;
}

// 노드의 구간 대표값: 기존의 구간 대표값 b1과, 새로운 값 b2를 병합
val_t update_b(val_t b1, val_t b2) {
    return b1 + b2;
}

pair<val_t, val_t> itree[size * 2];
pair<int, int> ptree[size * 2];

void init() {
    int i;
    for (i = 1 ; i < size * 2 ; i++)
        itree[i] = make_pair(init_value, init_value);
    for (i = size ; i < size * 2 ; i++)
        ptree[i] = make_pair(i, i);
    for (i = size - 1 ; i >= 1 ; i--)
        ptree[i] = make_pair(ptree[i << 1].first, ptree[i << 1 | 1].second);
}

void update(int s, int e, val_t val) { // [s, e]
    int s1, e1;
    int d = 0;
    s |= size;
    e |= size;
    s1 = s >> 1;
    e1 = e >> 1;
    while (s <= e) {
        if (s & 1) {
            itree[s].second = update_b(itree[s].second, val);
            val_t child_sum = sum(itree[s << 1].first, itree[s << 1 | 1].first);
            itree[s].first = update_a(
                (s >= size) ? init_value : child_sum,
                weight(itree[s].second, 1 << d));
        }
        if ((e & 1) == 0) {
            itree[e].second = update_b(itree[e].second, val);

            val_t child_sum = sum(itree[e << 1].first, itree[e << 1 | 1].first);
            itree[e].first = update_a(
                (e >= size) ? init_value : child_sum,
                weight(itree[e].second, 1 << d));
        }
        s = (s + 1) >> 1;
        e = (e - 1) >> 1;
        d++;
    }
    d = 1;
    while(s1) {
        itree[s1].first = update_a(
            sum(itree[s1 << 1].first, itree[s1 << 1 | 1].first),
            weight(itree[s1].second, 1 << d));
        itree[e1].first = update_a(
            sum(itree[e1 << 1].first, itree[e1 << 1 | 1].first),
            weight(itree[e1].second, 1 << d));
        s1 >>= 1;
        e1 >>= 1;
        d++;
    }
}

val_t _getrange2(int s, int e, int node) {
    if (node >= size)
        return itree[node].first;
    if (s <= ptree[node].first && e >= ptree[node].second)
        return itree[node].first;

    val_t cur = weight(itree[node].second,
        min(e, ptree[node].second) - max(s, ptree[node].first) + 1);
    int left = node << 1;
    int right = node << 1 | 1;

    if(s >= ptree[right].first)
        return update_a(_getrange2(s, e, right), cur);
    else if (e <= ptree[left].second)
```

```
      return update_a(_getrange2(s, e, left), cur);
   else
      return update_a(
         sum(_getrange2(s, e, left), _getrange2(s, e, right)),
         cur);
}

val_t getrange(int s, int e) { // [s, e]
   if(s > e) return init_value;
   return _getrange2(s | size, e | size, 1);
}

} // namespace itree
```

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;
}

} // namespace kmp
```

Suffix Array O(n log n)

```
#include <cstdio>
#include <algorithm>
using namespace std;

int n, K;
int dat[20003];

int ians[20003]; // ans -> index : 답의 반대
int ans[20003]; // index -> ans : 구하고자 하는 suffix array
int tmpans[20003]; // ans의 중간과정 저장

int bucket[20003]; // bucket -> index : starting points
int bucketcnt[20003]; // bucket -> count
int cntbucket; // number of buckets

int bucketmark[20003]; // ans -> bucket : 어느 bucket에 속하는가?
int bucketupdate[20003]; // ans -> bucketnumber. -1이면 새 거.

inline int sf(const int& a, const int& b) {
   return dat[a] < dat[b];
}

int main() {
   int i, H;
   scanf("%d%d", &n, &K);
   for (i = 0 ; i < n ; i++) {
      scanf("%d", &dat[i]);
      dat[i]++;
      ans[i] = i;
      ians[i] = i;
```

```
    }
    // constructing suffix array by doubling method
    // phase 1: init
    sort(ans, ans + n, sf);
    for (i = 0 ; i < n ; i++) {
        if (i == 0 || dat[ans[i]] != dat[ans[i - 1]]) {
            bucket[cntbucket] = i;
            bucketcnt[cntbucket] = 0;
            cntbucket++;
        }
        bucketmark[ans[i]] = cntbucket - 1;
    }
    // phase 2: doubling
    for (H = 1 ; ; H *= 2) {
        // phase 2-1: rearrangement
        // 현재 위치의 H만큼 뒤를 보면서 위치를 바꿈, 결과를 tmpans에 저장
        for (i = 0 ; i < n ; i++) {
            if (ans[i] >= n - H) {
                // 이 뒤는 null 문자이므로 앞으로 가야 한다.
                int tbuck = bucketmark[ans[i]];
                bucketupdate[ans[i]] = -1;
                tmpans[bucket[tbuck] + bucketcnt[tbuck]] = ans[i];
                bucketcnt[tbuck]++;
            }
        }
        for (i = 0 ; i < n ; i++) {
            if (ans[i] >= H) {
                // 위에서 처리하지 않은 나머지 것들
                int tbuck = bucketmark[ans[i] - H];
                bucketupdate[ans[i] - H] = bucketmark[ans[i]];
                tmpans[bucket[tbuck] + bucketcnt[tbuck]] = ans[i] - H;
                bucketcnt[tbuck]++;
            }
        }
/* 만약 정확히 길이가 K인 문자열 중 중복되는 것의 개수를 세려고 한다면,
 * 여기서 처리하라. 그래야 bucketmark가 H인 상태로 남아 있고
 * (bucketmark가 같으면 그 자리에서 H글자만큼의 문자열은 같다는 뜻)
 * 정렬은 2H 길이를 기준으로 되어 있으니까, tmpans를 이용하기.
 * 부분 문자열의 길이 K는 H 이상 2 * H 이하여야 함. */
        // phase 2-2: identify new buckets
        int lastbucket = bucketmark[tmpans[0]];
        for (i = 1 ; i < n ; i++) {
            if (bucket[bucketmark[tmpans[i]]] != i) {
                if (bucketupdate[tmpans[i]] != bucketupdate[tmpans[i - 1]]){
                    // found new bucket
                    bucket[cntbucket] = i;
                    lastbucket = cntbucket;
                    cntbucket++;
                }
            }
            else {
                lastbucket = bucketmark[tmpans[i]];
            }
            bucketmark[tmpans[i]] = lastbucket;
        }
        // phase 2-3: copy ans and calculate ians
        int flg = 0;
        bucketmark[n] = -1;
        for (i = 0 ; i < n ; i++) {
            if(bucketmark[tmpans[i]] == bucketmark[tmpans[i + 1]]) flg = 1;
            ans[i] = tmpans[i];
            ians[ans[i]] = i;
            bucketcnt[bucketmark[ans[i]]] = 0;
        }
        if (flg == 0) break;
    }
    return 0;
}
```

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(i, i + 1);
    return 0;
}
```

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 없이 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.