

University of Haifa ICPC Team Notebook 2017

Contents

1 Miscellaneous

1.1	Template	1
1.2	Input/Output c++	1
1.3	Random STL stuff	1
1.4	2-SAT Solver, SCC, T-Sort	2
1.5	Knuth-Morris-Pratt	2
1.6	Longest Increasing Subsequence	3
1.7	Longest Common Subsequence	3
1.8	Dates Python	3
1.9	Dates C++	3

2 Geometry

2.1	Geometry Vectors	4
2.2	Convex Hull	4
2.3	Minimal Bounding Circle	4

3 Numerical algorithms

3.1	Gauss Elimination	5
3.2	Fast Fourier Transform	5
3.3	Hadamard Transform	6
3.4	Fast Subset Convolution	7
3.5	Number Theory	7
3.6	More Number Theory	8
3.7	Monotone Recursion (Template)	9

4 Graph algorithms

4.1	Dijkstra	11
4.2	Dinic Max Flow	11
4.3	Maximum Bipartite Matching	11
4.4	Dominators	12
4.5	Lowest Common Ancestor	13
4.6	All Pairs Shortest Paths (Floyd Warshall)	13

5 Data structures

5.1	Disjoint Sets (Union-Find)	14
5.2	Fenwick Tree	14
5.3	Segment Tree	14
5.4	Convex Hull Trick	15
5.5	Linear Convex Hull Trick	15

1 Miscellaneous

1.1 Template

```
#include <bits/stdc++.h>
#define loop(i, a, n) for (int i = a; i < int(n); ++i)
#define loop_rev(i, b, a) for (ll i = b; i >= ll(a); --i)
using namespace std;
```

```
//optional:
#define int ll
```

```
typedef long long ll;
typedef long double ld;
typedef pair<int,int> ii;
typedef vector<int> vi;
```

```
typedef vector<vi> vvi;
```

```
void solve() {
}
```

```
int32_t main() {
    int t; cin >> t;
    while (t--)
        solve();
    return 0;
}
```

1.2 Input/Output c++

```
int main()
{
    // Ouput a specific number of digits past the decimal point,
    // in this case 5
    cout.setf(ios::fixed); cout << setprecision(5);
    cout << 100.0/7.0 << endl;
    cout.unsetf(ios::fixed);

    // Output the decimal point and trailing zeros
    cout.setf(ios::showpoint);
    cout << 100.0 << endl;
    cout.unsetf(ios::showpoint);

    // Output a '+' before positive values
    cout.setf(ios::showpos);
    cout << 100 << " " << -100 << endl;
    cout.unsetf(ios::showpos);

    // Output numerical values in hexadecimal
    cout << hex << 100 << " " << 1000 << " " << 10000 << dec << endl;

    //print 6 digits after the point
    printf("%.6f", (float)0.2426353);
    cout.setprecision(6);
    cout << fixed << 0.2426353 << endl;
}
```

1.3 Random STL stuff

```
// Example for using stringstream and next_permutation
```

```
int main(void) {
    vector<int> v;

    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    v.push_back(4);
}
```

```

// Expected output: 1 2 3 4
//                  1 2 4 3
//                  ...
//                  4 3 2 1
do {
    ostringstream oss;
    oss << v[0] << " " << v[1] << " " << v[2] << " " << v[3];

    // for input from a string s,
    // istringstream iss(s);
    //  iss >> variable;

    cout << oss.str() << endl;
} while (next_permutation (v.begin(), v.end()));

v.clear();

v.push_back(1);
v.push_back(2);
v.push_back(1);
v.push_back(3);

// To use unique, first sort numbers. Then call
// unique to place all the unique elements at the beginning
// of the vector, and then use erase to remove the duplicate
// elements.

sort(v.begin(), v.end());
v.erase(unique(v.begin(), v.end()), v.end());

// Expected output: 1 2 3
for (size_t i = 0; i < v.size(); i++)
    cout << v[i] << " ";
cout << endl;
}

```

1.4 2-SAT Solver, SCC, T-Sort

```

void scc_dfs(const graph& G, int v, vector<int>& visit, vector<int>&
stack){
    if (visit[v]) return;

    visit[v] = true;
    for (auto& x : G[v]) scc_dfs(G, x, visit, stack);
    stack.push_back(v);
}

void scc_dfs(const graph& G, int v, int s, vector<int>& visit, vector<
int>& res){
    if (visit[v]) return;

    visit[v] = true;
    res[v] = s;

```

```

    for (auto& x : G[v]) scc_dfs(G, x, s, visit, res);
}

inline vector<int> scc_t_sort(const graph& G){
    graph G_rev(G.size());
    loop(i, 0, G.size()) for (auto& j : G[i]) G_rev[j].push_back(i);
    vector<int> visit(G.size(), false), stack;
    loop(i, 0, G.size()) scc_dfs(G, i, visit, stack);
    visit.assign(G.size(), false);
    vector<int> ret(G.size());
    int counter = 0;
    while (stack.size()) scc_dfs(G_rev, stack.back(), counter++, visit
, ret), stack.pop_back();
    return ret;
}

inline void add_or(graph& G, int a, int b, const vector<int>& not_v){
    G[not_v[a]].push_back(b);
    G[not_v[b]].push_back(a);
}

vector<int> sat_2(const vector<ii>& or_c, const vector<int>& not_v){
    graph G(not_v.size());
    for (auto& x : or_c) add_or(G, x.first, x.second, not_v);
    vector<int> arr = scc_t_sort(G);
    loop(i, 0, arr.size()) if (arr[i] == arr[not_v[i]]) return {};
    vector<int> val(G.size());
    loop(i, 0, arr.size()) if (arr[i] > arr[not_v[i]]) val[i] = true,
        val[not_v[i]] = false;
    return val;
}

```

1.5 Knuth-Morris-Pratt

```

vector<int> kmp(string T, string P){
    int n = T.size();
    int m = P.size();
    vector<int> prefix(m, -1);
    int j = -1;
    for(int i = 1; i < m; i++){
        while(j != -1 && P[i] != P[j+1])
            j = prefix[j];
        if(P[i] == P[j+1])
            j++;
        prefix[i] = j;
    }
    j = -1;
    vector<int> pos;
    for(int i = 0; i < n; i++){
        while(j != -1 && T[i] != P[j+1])
            j = prefix[j];
        if(T[i] == P[j+1])
            j++;
    }
}

```

```

        if(j == m-1)
            pos.push_back(i), j = prefix[j];
    }
    return pos;
}

```

1.6 Longest Increasing Subsequence

```

vector<int> LIS(vector<int> arr, bool strict = true){
    static const int inf = 2e9;
    vector<int> best(arr.size(), arr.size());
    arr.push_back(inf);
    vector<int> back(arr.size(), -1);

#define line(func) j = func(best.begin(), best.end(), i, [&arr](
    int a, int b){return arr[a] < arr[b];}) - best.begin()

    loop(i, 0, arr.size() - 1){
        int j;
        if (strict) line(lower_bound);
        else line(upper_bound);
        if (j != 0) back[i] = best[j - 1];
        best[j] = i;
    }

#undef line

    int pos = 0;
    while (pos < int(best.size()) - 1 && best[pos + 1] != int(arr.size()
        - 1) ++pos;
    pos = best[pos];
    vector<int> ret;
    while (pos != -1) ret.push_back(pos), pos = back[pos];
    reverse(ret.begin(), ret.end());
    return ret;
}

```

1.7 Longest Common Subsequence

```

vector<char> LCS(string a, string b){
    int n = a.size(), m = b.size();
    vector<vector<int>> > dp(n, vector<int>(m));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            if(a[i] == b[j])
                dp[i][j] = (j && i)? dp[i-1][j-1]+1 : 1;
            else
                dp[i][j] = max(j? dp[i][j-1] : 0, i? dp[i-1][j] : 0);
        }
    }
    stack<char> ans;
    for(int i = n-1, j = m-1; i >= 0 && j >= 0;){
        if(a[i] == b[j]){

```

```

            ans.push(a[i]);
            i--; j--;
        }
        else{
            if((i? dp[i-1][j] : 0) < (j? dp[i][j-1] : 0))
                j--;
            else
                i--;
        }
    }
    vector<char> res;
    while(!ans.empty())
        res.push_back(ans.top()), ans.pop();
    return res;
}

```

1.8 Dates Python

```

import datetime

jdate = 17330
date = datetime.datetime.strptime(str(jdate), '%y%j').date()
# date is (date.day, date.month, date.year)

date = datetime.date(2017, 11, 26)

jdate = int( date.strftime('%y%j') )

```

1.9 Dates C++

// Routines for performing computations on dates. In these routines, months are expressed as integers from 1 to 12, days are expressed as integers from 1 to 31, and years are expressed as 4-digit integers.

```

#include <iostream>
#include <string>

using namespace std;

string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

// converts Gregorian date to integer (Julian day number)
int dateToInt (int m, int d, int y){
    return
        1461 * (y + 4800 + (m - 14) / 12) / 4 +
        367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
        3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
        d - 32075;
}

```

```
// converts integer (Julian day number) to Gregorian date: month/day/
year
void intToDate (int jd, int &m, int &d, int &y){
    int x, n, i, j;

    x = jd + 68569;
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string intToDay (int jd){
    return dayOfWeek[jd % 7];
}

int main (int argc, char **argv){
    int jd = dateToInt (3, 24, 2004);
    int m, d, y;
    intToDate (jd, m, d, y);
    string day = intToDay (jd);

    // expected output:
    //      2453089
    //      3/24/2004
    //      Wed
    cout << jd << endl
         << m << "/" << d << "/" << y << endl
         << day << endl;
}
```

2 Geometry

2.1 Geometry Vectors

```
typedef double T;
typedef pair<T, T> ii;
typedef pair<ii, T> i3;
typedef pair<ii, ii> i4;

inline double cross(ii a, ii b){
    return a.first * b.y - a.y * b.first;
}

inline ii operator+(ii a, ii b){
    return ii(a.first + b.first, a.y + b.y);
}
```

```
inline ii operator-(ii a, ii b){
    return ii(a.first - b.first, a.y - b.y);
}

inline double operator*(ii a, ii b){
    return a.first * b.first + a.y * b.y;
}

inline ii operator/(ii a, double b){
    return ii(a.first / b, a.y / b);
}

inline T norm(ii x){
    return x * x;
}

inline int sig(double x){
    return x > 0 ? 1 : x == 0 ? 0 : -1;
}

inline double distance(ii a, ii b){
    return sqrt(norm(b - a));
}

bool inTriangle(ii a, ii b, ii c, ii p){
    int sign1 = sign(cross(a-p, b-a));
    int sign2 = sign(cross(b-p, c-b));
    int sign3 = sign(cross(c-p, a-c));
    if(max(sign1, max(sign2, sign3)) == 1 && min(sign1, min(sign2, sign3))
       == -1)
        return false;
    return true;
}

ll area(vector<ii> shape){ // counter-clockwise
    ll sum = 0;
    while (shape.size() > 2){
        sum += cross(shape[shape.size() - 2] - shape[0], shape[shape.
            size() - 1] - shape[0]);
        shape.pop_back();
    }
    return sum;
}
```

2.2 Convex Hull

```
inline int lowest_point(vector<ii> P){
    return min_element(P.begin(), P.end(), [](ii a, ii b){return
        ii(a.second, a.first) < ii(b.second, b.first);}) - P.begin
        ();
}

vector<ii> convex_hull(vector<ii> S){
```

```

int first = lowest_point(S);
ii origin = S[first];
S.erase(S.begin() + first);
//sort by angle
sort(S.begin(), S.end(), [&origin](ii p1, ii p2){return ii(-cross(
    p1 - origin, p2 - origin), norm(p1 - origin)) < ii(0, norm(p2
    - origin));});

vector<ii> hull = {origin};
for (auto& p : S){
    while(hull.size() >= 2 && cross(hull[hull.size() - 1] - hull[
        hull.size() - 2], p - hull[hull.size() - 1]) <= 0) hull.
        pop_back();
    hull.push_back(p);
}
return hull;
}

```

2.3 Minimal Bounding Circle

```

bool inside_circle(i3 circle, ii p){
    return distance(circle.first, p) <= circle.second;
}

i3 circle_from_2_points(ii a, ii b){
    return {(a+b)/2, distance(a, b)/2};
}

i3 circle_from_3_points(ii a, ii b, ii c){
    double d = 2.0*(a.x*(b.y - c.y) + b.x*(c.y - a.y) + c.x*(a.y - b.y
    ));
    double xc = ((a.x * a.x + a.y * a.y) * (b.y - c.y) + (b.first * b.
        first + b.y * b.y) * (c.y - a.y) + (c.first * c.first + c.y *
        c.y) * (a.y - b.y) ) / d;
    double yc = ((a.first * a.first + a.y * a.y) * (c.first - b.first)
        + (b.first * b.first + b.y * b.y) * (a.first - c.first) + (c.
        first * c.first + c.y * c.y) * (b.first - a.first) ) / d;
    cerr << xc << " " << yc << endl;
    ii center = {xc, yc};
    return {center, distance(center, a)};
}

i3 minimal_bounding_circle_3(vector<ii>& P, ii a, ii b){
    i3 circle = circle_from_2_points(a, b);
    for(ii p : P){
        if(!inside_circle(circle, p)){
            circle = circle_from_3_points(a, b, p);
        }
    }
    return circle;
}

i3 minimal_bounding_circle_2(vector<ii>& P, ii a){

```

```

i3 circle = circle_from_2_points(a, P[0]);
vector<ii> P2 = {P[0]};
for(int i = 1; i < (int)P.size(); ++i){
    ii p = P[i];
    if(!inside_circle(circle, p)){
        circle = minimal_bounding_circle_3(P2, a, p);
    }
    P2.push_back(P[i]);
}
return circle;
}

i3 minimal_bounding_circle(vector<ii>& P){
    if(P.size() < 2) return {{0,0}, -1}; // null!
    random_shuffle(P.begin(), P.end());

    i3 circle = circle_from_2_points(P[0], P[1]);
    vector<ii> P2 = {P[0], P[1]};
    for(int i = 2; i < (int)P.size(); ++i){
        ii p = P[i];
        if(!inside_circle(circle, p)){
            circle = minimal_bounding_circle_2(P2, p);
        }
        P2.push_back(P[i]);
    }
    return circle;
}

```

3 Numerical algorithms

3.1 Gauss Elimination

```

const int mod = 2;

inline int mul(int a, int b){
    return (ll(a) * ll(b)) % mod;
}

int power(int a, int b){
    if (b == 0) return 1;
    int c = power(a, b / 2);
    return mul(mul(c, c), (b % 2 == 0 ? 1 : a));
}

inline int inv(int x){
    return power(x, mod - 2);
}

vector<int> operator-(vector<int> arr1, const vector<int>& arr2){
    loop(i, 0, arr1.size()) arr1[i] = (ll(arr1[i]) - ll(arr2[i]) + mod
        ) % mod;
    return arr1;
}

```

```

vector<int> operator*(int k, vector<int> arr){
    for (auto& x : arr) x = (ll(x) * ll(k)) % mod;
    return arr;
}

#define N 2501
#define bool_array bitset<N>
const bool_array zero;

template<typename T>
void gauss(vector<T>& arr);

bool_array operator-(const bool_array& arr1, const bool_array& arr2){
    return arr1 ^ arr2;
}

bool_array operator*(int k, const bool_array& arr){
    return k == 1 ? arr : zero;
}

bool_array vector_int_to_bool_array(const vector<int>& arr){
    bool_array ret;
    loop(i, 0, arr.size()) ret[i] = arr[i];
    return ret;
}

vector<int> bool_array_to_vector_int(const bool_array& arr, int n){
    vector<int> ret(n);
    loop(i, 0, ret.size()) ret[i] = arr[i];
    return ret;
}

void gauss_com(vector<vector<int>>& arr){
    int m = arr.front().size();
    vector<bool_array> com(arr.size());
    loop(i, 0, com.size()) com[i] = vector_int_to_bool_array(arr[i]);
    gauss(com);
    arr.resize(com.size());
    loop(i, 0, com.size()) arr[i] = bool_array_to_vector_int(com[i], m);
}

template<typename T>
void gauss(vector<T>& arr){
    int row = 0;
    if (arr.size() == 0) return;
    loop(j, 0, arr[row].size()){
        int pos = -1;
        loop(k, row, arr.size()) if (arr[k][j] != 0){
            pos = k;
            break;
        }
        if (pos == -1) continue;
        swap(arr[pos], arr[row]);

```

```

        loop(k, row + 1, arr.size()) if (arr[k][j] != 0) arr[k] = arr[
            row][j] * arr[k] - arr[k][j] * arr[row];
        if (++row == (int)arr.size()) break;
    }

    while (arr.size() != 0){
        int b = true;
        loop(j, 0, arr.back().size()) if (arr.back()[j] != 0){
            b = false;
            break;
        }
        if (!b) break;
        arr.pop_back();
    }
    int col = -1;
    loop(i, 0, arr.size()){
        while (arr[i][++col] == 0);
        loop(k, 0, i) if (arr[k][col] != 0) arr[k] = arr[i][col] * arr[
            k] - arr[k][col] * arr[i];
    }
}

```

3.2 Fast Fourier Transform

```

#define pi 3.14159265359

typedef complex<double> com;

void dft(vector<com>& p, com mult){
    if (p.size() == 1) return;

    vector<com> p1(p.size() / 2), p2(p.size() / 2);
    loop(i, 0, p.size())
        if (i % 2 == 0) p1[i / 2] = p[i];
        else p2[i / 2] = p[i];

    com curr = 1, new_mult = mult * mult;
    dft(p1, new_mult), dft(p2, new_mult);
    loop(i, 0, p.size() / 2){
        com a = p1[i], b = curr * p2[i];
        p[i] = a + b, p[i + p.size() / 2] = a - b;
        curr *= mult;
    }
}

void dft(vector<com>& p, int k){
    dft(p, polar(1., k * 2 * pi / p.size()));
}

vector<double> mul(const vector<double>& _p1, const vector<double>&
    _p2){
    vector<com> p1(_p1.size()), p2(_p2.size());
    loop(i, 0, p1.size()) p1[i] = _p1[i];
    loop(i, 0, p2.size()) p2[i] = _p2[i];

```

```

int k = 1;
while (k < 2 * (int)p1.size() - 1 || k < 2 * (int)p2.size() - 1) k
    *= 2;
while ((int)p1.size() < k) p1.push_back(0);
while ((int)p2.size() < k) p2.push_back(0);

dft(p1, 1), dft(p2, 1);

vector<com> p(p1.size());
loop(i, 0, p1.size()) p[i] = p1[i] * p2[i];

dft(p, -1);

while (p.size() > 1 && norm(p.back()) < 0.001) p.pop_back();
vector<double> res(p.size());
loop(i, 0, res.size()) res[i] = real(p[i]) / p1.size();
return res;
}

```

3.3 Hadamard Transform

```

//
//Given A[], B[], find C[] such that
//  C[i] = sum_j A[j] B[i xor j]
//

const unsigned int mod = 1'000'000'007;

void trans(vi::iterator begin, vi::iterator end, int counter){
    if (counter == 0) return;
    int k = (end - begin) / 2;

    trans(begin, begin + k, counter - 1), trans(begin + k, end,
        counter - 1);
    loop(i, 0, k){
        unsigned int x = *(begin + i), y = *(begin + k + i);
        *(begin + i) = (x + y) % mod, *(begin + k + i) = (x + mod - y)
            % mod;
    }
}

vi mul(vi p1, vi p2, bool same = false){
    int k, counter;
    for (k = 1, counter = 0; k < (int)max(p1.size(), p2.size()); k *=
        2, ++counter);
    p1.resize(k), p2.resize(k);

    trans(p1.begin(), p1.end(), counter);
    if (!same) trans(p2.begin(), p2.end(), counter);
    else p2 = p1;
    loop(i, 0, p1.size()) p1[i] = (ll(p1[i]) * ll(p2[i])) % mod;
    trans(p1.begin(), p1.end(), counter);
}

```

```

int curr = 1, mul = mod / 2 + 1;
loop(i, 0, counter) curr = (ll(curr) * ll(mul)) % mod;
for (auto& x : p1) x = (ll(x) * ll(curr)) % mod;
return p1;
}

vi power(vi p, int k){
    vi m = p;
    p.assign(p.size(), 0); p[0] = 1;
    while (k){
        if (k % 2 == 1) mul(p, m);
        k /= 2;
        mul(m, m, true);
    }
    return p;
}

int32_t main(){
    ios::sync_with_stdio(false);
    int k, l; cin >> k >> l;
    vi p(l + 1, 1);
    p[0] = p[1] = 0;
    loop(i, 2, p.size()) for (int j = 2 * i; j < (int)p.size(); j += i
        ) p[j] = 0;

    p = power(p, k);
    cout << p[0] << endl;
    return 0;
}

```

3.4 Fast Subset Convolution

```

const unsigned int mod = 1'000'000'007;

void trans(vi& p){
    int n = p.size();
    for(int c = 1; c < n; c+=c)
        loop(i, 0, n)
            if(i & c)
                p[i] += p[i-c];
}

void invtrans(vi& p){
    int n = p.size();
    for(int c = n/2; c; c /=2)
        for(int i = n-1; i>= 0; i--)
            if(i & c)
                p[i] -= p[i-c];
}

vi mul(vi p1, vi p2){
    int n = p1.size();

    //loop(j, 0, n) cerr << p1[j]; cerr << endl;
}

```

3.5 Number Theory

// Not Tested

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

// return a % b (positive value)

```
int mod(int a, int b) {
    return ((a%b) + b) % b;
}
```

// computes gcd(a,b)

```
int gcd(int a, int b) {
    while (b) { int t = a%b; a = b; b = t; }
    return a;
}
```

// computes lcm(a,b)

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

// (a^b) mod m via successive squaring

```
int powermod(int a, int b, int m)
{
    int ret = 1;
    while (b)
    {
        if (b & 1) ret = mod(ret*a, m);
        a = mod(a*a, m);
        b >>= 1;
    }
    return ret;
}
```

// returns g = gcd(a, b); finds x, y such that d = ax + by

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

// finds all solutions to ax = b (mod n)

```
vi modular_linear_equation_solver(int a, int b, int n) {
    int x, y;
    vi ret;
```

```
//loop(j, 0, n) cerr << p2[j]; cerr << endl;
//cerr << endl;
trans(p1);
//loop(j, 0, n) cerr << p1[j]; cerr << endl;
trans(p2);
//loop(j, 0, n) cerr << p2[j]; cerr << endl;

loop(i, 0, n) p1[i] *= p2[i];

invtrans(p1);

//loop(j, 0, n) cerr << p1[j]; cerr << endl;
//cerr << endl;
return p1;
}

int32_t main(){
    ios::sync_with_stdio(false);

    int n, m, k; cin >> n >> m >> k;
    vvi A(m, vi(1 << n));
    loop(i, 0, m)
    loop(j, 0, (1 << n)){
        char c; cin >> c;
        A[i][j] = (c == '1');
    }
    loop(i, 0, k){
        int a,b ; cin >> a >> b;
        vi ans = mul(A[a], A[b]);
        loop(j, 0, (1 << n))
            cout << (ans[j] ? 1 : 0);
        cout << endl;
    }

    return 0;
}

/*
3 5 3
11111010
11000000
11001000
11101000
11101000
0 1
1 2
3 4

*/
```



```

int g = extended_euclid(a, n, x, y);
if (!(b%g)) {
    x = mod(x*(b / g), n);
    for (int i = 0; i < g; i++)
        ret.push_back(mod(x + i*(n / g), n));
}
return ret;
}

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

// Chinese remainder theorem (special case): find z such that
// z % m1 = r1, z % m2 = r2. Here, z is unique modulo M = lcm(m1, m2)
.
// Return (z, M). On failure, M = -1.
ii chinese_remainder_theorem(int m1, int r1, int m2, int r2) {
    int s, t;
    int g = extended_euclid(m1, m2, s, t);
    if (r1%g != r2%g) return make_pair(0, -1);
    return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2) / g, m1*m2 / g)
    ;
}

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

// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x, int &y) {
    if (!a && !b)
    {
        if (c) return false;
        x = 0; y = 0;
        return true;
    }
    if (!a)
    {

```

```

        if (c % b) return false;
        x = 0; y = c / b;
        return true;
    }
    if (!b)
    {
        if (c % a) return false;
        x = c / a; y = 0;
        return true;
    }
    int g = gcd(a, b);
    if (c % g) return false;
    x = c / g * mod_inverse(a / g, b / g);
    y = (c - a*x) / b;
    return true;
}

int main() {
    // expected: 2
    cout << gcd(14, 30) << endl;

    // expected: 2 -2 1
    int x, y;
    int g = extended_euclid(14, 30, x, y);
    cout << g << " " << x << " " << y << endl;

    // expected: 95 451
    vi sols = modular_linear_equation_solver(14, 30, 100);
    for (int i = 0; i < sols.size(); i++) cout << sols[i] << " ";
    cout << endl;

    // expected: 8
    cout << mod_inverse(8, 9) << endl;

    // expected: 23 105
    //          11 12
    ii ret = chinese_remainder_theorem(vi({ 3, 5, 7 }), vi({ 2, 3,
        2 }));
    cout << ret.first << " " << ret.second << endl;
    ret = chinese_remainder_theorem(vi({ 4, 6 }), vi({ 3, 5 }));
    cout << ret.first << " " << ret.second << endl;

    // expected: 5 -15
    if (!linear_diophantine(7, 2, 5, x, y)) cout << "ERROR" <<
        endl;
    cout << x << " " << y << endl;
    return 0;
}

```

3.6 More Number Theory

// Not Tested

```
int max(int a, int b)
```

```

{
    return a>b ? a:b;
}

int min(int a, int b)
{
    return a<b ? a:b;
}

int gcd(int a, int b)
{
    if (b==0) return a;
    else return gcd(b, a%b);
}

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

bool prime(int n)
{
    if (n<2) return false;
    for (int i=2;i<=n;i++)
        if (n%i==0) return false;
    return true;
}

bool isLeap(int n)
{
    if (n%100==0)
        if (n%400==0) return true;
        else return false;

    if (n%4==0) return true;
    else return false;
}

long powmod(long base, long exp, long modulus) {
    base %= modulus;
    long result = 1;
    while (exp > 0) {
        if (exp & 1) result = (result * base) % modulus;
        base = (base * base) % modulus;
        exp >>= 1;
    }
    return result;
}

int factmod (int n, int p) {
    long long res = 1;
    while (n > 1) {
        res = (res * powmod (p-1, n/p, p)) % p;
        for (int i=2; i<=n%p; ++i)
            res=(res*i) %p;
    }
}

```

```

        n /= p;
    }
    return int (res % p);
}

void combination(int n, int m)
{
    if (n<m) return ;
    int a[50]={0};
    int k=0;

    for (int i=1;i<=m;i++) a[i]=i;
    while (true)
    {
        for (int i=1;i<=m;i++)
            cout << a[i] << " ";
        cout << endl;

        k=m;
        while ((k>0) && (n-a[k]==m-k)) k--;
        if (k==0) break;
        a[k]++;
        for (int i=k+1;i<=m;i++)
            a[i]=a[i-1]+1;
    }
}

int main(void)
{
    /***** Max or min *****/

    cout << "-----TEST FOR MAX OR MIN-----" << endl;
    // Max and min are implemented in library
    // the point for this is to show the syntax

    cout << "Max of (5,7): " << max(5,7) << endl;
    cout << "Min of (5,7): " << min(5,7) << endl;

    cout << "-----TEST FOR MAX OR MIN-----" << endl;
    cout << endl << endl;

    /***** GCD and LCM *****/

    cout << "-----TEST FOR GCD AND LCM-----" << endl;

    cout << "GCD of (12, 15): " << gcd(12,15) << endl;
    cout << "LCM of (12, 15): " << lcm(12,15) << endl;

    cout << "-----TEST FOR GCD AND LCM-----" << endl;
    cout << endl << endl;

    /***** prime number *****/

    cout << "-----TEST FOR PRIME NUMBER-----" << endl;
}

```

```

cout << "Is 1251 prime?: " << prime(1251) << endl;
cout << "Is 97 prime? : " << prime(97) << endl;

cout << "-----TEST FOR PRIME NUMBER-----" << endl;
cout << endl << endl;

/***** Leap year *****/

cout << "-----TEST FOR LEAP YEAR-----" << endl;

cout << "2012 is Leap? : " << isLeap(2012) << endl;
cout << "1900 is Leap? : " << isLeap(1900) << endl;
cout << "1903 is Leap? : " << isLeap(1903) << endl;

cout << "-----TEST FOR LEAP YEAR-----" << endl;
cout << endl << endl;

/***** a^b mod p and n! mod p *****/

cout << "-----TEST FOR (A^B MOD P) AND (N! MOD P)-----" <<
    endl;

cout << "5^6 mod 17: " << powmod(5, 6, 17) << endl;
cout << "17 mod 17 : " << factmod(17, 17) << endl;

cout << "-----TEST FOR (A^B MOD P) AND (N! MOD P)-----" <<
    endl;
cout << endl << endl;

/***** Generate combinations *****/

cout << "-----TEST FOR GENERATING COMBINATIONS-----" << endl
    ;

combination(6, 3); // pick 3 numbers from 6 numbers

cout << "-----TEST FOR GENERATING COMBINATIONS-----" << endl
    ;
cout << endl << endl;

return 0;
}

```

3.7 Monotone Recursion (Template)

```

void rec(int n1, int n2, int m1, int m2, DSU dsu){
    if(n2 < n1 || m2 < m1) return;
    int mid = (n1 + n2) / 2;
    DSU dsu2 = dsu;
    for(int i = mid; i <= min(n2, m1-1); i++){
        dsu.uni(E[i].u, E[i].v);
    }
    DSU dsu1 = dsu;
    int k = max(m1-1, mid-1);

```

```

while(k < m2 && !dsu.single()){
    if (k > n2) dsu2.uni(E[k].u, E[k].v);
    ++k;
    dsu.uni(E[k].u, E[k].v);
}

if(dsu.single()){
    ans[mid] = k;
    rec(n1, mid-1, m1, k, dsu1);
    rec(mid+1, n2, k, m2, dsu2);
}
else{
    rec(n1, mid-1, m1, m2, dsu1);
}
}

```

4 Graph algorithms

4.1 Dijkstra

```

vector<int> dijkstra(const graph_w& G, int s){
    int n = G.size();
    vector<int> dis(n, inf);
    set<ii> S;
    dis[s] = 0;
    S.insert({0, s});
    while(!S.empty()){
        int u = S.begin()->second;
        S.erase(S.begin());

        for(auto& e : G[u]){
            int v = e.first, w = e.second;
            if(dis[v] > dis[u] + w){
                S.erase({dis[v], v});
                dis[v] = dis[u] + w;
                S.insert({dis[v], v});
            }
        }
    }
    return dis;
}

```

4.2 Dinic Max Flow

```

//
// Dinic algorithm for maximum flow / minimum cut
// time: O(VVE), usually faster, no more than O(maxflow * E)
// space: O(V+E)
//
struct edge {
    int u, v;
    ll cap, flow;

```

```

edge() {}
edge(int u, int v, ll cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic {
    int N;
    vector<edge> E;
    vector<vector<int>> > g;
    vector<int> d, pt;

    Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

    void Addedge(int u, int v, ll cap) {
        if (u != v) {
            E.emplace_back(edge(u, v, cap));
            g[u].emplace_back(E.size() - 1);
            E.emplace_back(edge(v, u, 0));
            g[v].emplace_back(E.size() - 1);
        }
    }

    bool BFS(int S, int T) {
        queue<int> q({S});
        fill(d.begin(), d.end(), N + 1);
        d[S] = 0;
        while(!q.empty()) {
            int u = q.front(); q.pop();
            if (u == T) break;
            for (int k: g[u]) {
                edge &e = E[k];
                if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
                    d[e.v] = d[e.u] + 1;
                    q.emplace(e.v);
                }
            }
        }
        return d[T] != N + 1;
    }

    ll DFS(int u, int T, ll flow = -1) {
        if (u == T || flow == 0) return flow;
        for (int &i = pt[u]; i < g[u].size(); ++i) {
            edge &e = E[g[u][i]];
            edge &oe = E[g[u][i]^1];
            if (d[e.v] == d[e.u] + 1) {
                ll amt = e.cap - e.flow;
                if (flow != -1 && amt > flow) amt = flow;
                if (ll pushed = DFS(e.v, T, amt)) {
                    e.flow += pushed;
                    oe.flow -= pushed;
                    return pushed;
                }
            }
        }
        return 0;
    }
};

```

```

}

ll MaxFlow(int S, int T) {
    ll total = 0;
    while (BFS(S, T)) {
        fill(pt.begin(), pt.end(), 0);
        while (ll flow = DFS(S, T))
            total += flow;
    }
    return total;
}
};

```

4.3 Maximum Bipartite Matching

```

//Not Tested

// This code performs maximum bipartite matching.
//
// Running time:  $O(|E| |V|)$  -- often much faster in practice
//
// INPUT: w[i][j] = edge between row node i and column node j
// OUTPUT: mr[i] = assignment for row node i, -1 if unassigned
//         mc[j] = assignment for column node j, -1 if unassigned
//         function returns number of matches made

bool find_match(int i, const vvi &w, vi &mr, vi &mc, vi &seen) {
    for (int j = 0; j < w[i].size(); j++) {
        if (w[i][j] && !seen[j]) {
            seen[j] = true;
            if (mc[j] < 0 || find_match(mc[j], w, mr, mc, seen)) {
                mr[i] = j;
                mc[j] = i;
                return true;
            }
        }
    }
    return false;
}

int bipartite_matching(const vvi &w, vi &mr, vi &mc) {
    mr = vi(w.size(), -1);
    mc = vi(w[0].size(), -1);

    int ct = 0;
    for (int i = 0; i < w.size(); i++) {
        vi seen(w[0].size());
        if (find_match(i, w, mr, mc, seen)) ct++;
    }
    return ct;
}

```

4.4 Dominators

```
namespace dominator{
    int T, n;
    vvi g, rg, bucket;
    vi sdom, par, dom, dsu, label, arr, rev;

    int find(int u, int x = 0){
        if (u == dsu[u]) return x ? -1 : u;
        int v = find(dsu[u], x + 1);
        if (v < 0) return u;
        if (sdom[label[dsu[u]]] < sdom[label[u]]) label[u] = label[dsu[u]];
        dsu[u] = v;
        return x ? v : label[u];
    }

    void unite(int u, int v){
        dsu[v] = u;
    }

    void dfs(int u){
        T++; arr[u] = T; rev[T] = u;
        label[T] = T; sdom[T] = T; dsu[T] = T;
        loop(i, 0, g[u].size()){
            int w = g[u][i];
            if (!arr[w]) dfs(w), par[arr[w]] = arr[u];
            rg[arr[w]].PB(arr[u]);
        }
    }

    static vi get(const graph& G, int root){
        T = 0, n = G.size();
        int k = n + 1;
        g.assign(k, {}), rg.assign(k, {}), bucket.assign(k, {});
        sdom.assign(k, 0), par.assign(k, 0), dom.assign(k, 0), dsu.
            assign(k, 0), label.assign(k, 0), arr.assign(k, 0), rev.
            assign(k, 0);

        loop(i, 0, G.size()) for (auto& x : G[i]) g[i + 1].PB(x + 1);
        ++root;
        vector<int> res(n, -1);
        dfs(root); n = T;
        loop_rev(i, n, 1){
            loop(j, 0, rg[i].size()) sdom[i] = min(sdom[i], sdom[find(
                rg[i][j]]));
            if (i > 1) bucket[sdom[i]].PB(i);
            loop(j, 0, bucket[i].size()){
                int w = bucket[i][j], v = find(w);
                if (sdom[v] == sdom[w]) dom[w] = sdom[w];
                else dom[w] = v;
            }
            if (i > 1) unite(par[i], i);
        }
    }
}
```

```
        loop(i, 2, n + 1){
            if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
            res[rev[i] - 1] = rev[dom[i]] - 1;
        }
        return res;
    }
}
```

4.5 Lowest Common Ancestor

```
//maybe remove

int LOGN;
vector<vector<int>> > LCA;
vector<int> depth;
vector<vector<int>> > tree;
void dfs(int u, int dep){
    depth[u] = dep;
    for(int v : tree[u])
        dfs(v, dep+1);
}

void init(int N, int fathers[]){
    tree.assign(N, vector<int>());
    int root = -1;
    for(int i = 0; i < N; i++){
        if(fathers[i] == -1)
            root = i, fathers[i] = i;
        else
            tree[fathers[i]].push_back(i);
    }
    depth.assign(N, 0);
    dfs(root, 0);
    LOGN = 3;
    for(int n = N; n /= 2; LOGN++);
    LCA.assign(LOGN, vector<int>(N));
    for(int i = 0; i < N; i++)
        LCA[0][i] = fathers[i];
    for(int d = 1; d < LOGN; d++)
        for(int i = 0; i < N; i++)
            LCA[d][i] = LCA[d-1][LCA[d-1][i]];
}

int query(int u, int v){
    if(depth[u] < depth[v])
        swap(u, v);
    for(int d = LOGN-1; d >= 0; d--)
        if(depth[LCA[d][u]] >= depth[v])
            u = LCA[d][u];
    if(u == v)
        return u;
    for(int d = LOGN-1; d >= 0; d--)
        if(LCA[d][u] != LCA[d][v])
            u = LCA[d][u], v = LCA[d][v];
    return LCA[0][u];
}
```

```

}

int main()
{
    int t; cin >> t;
    for(int r = 0; r<t; r++){
        int n; cin >> n;
        int *fathers = new int[n];
        for(int i = 0; i<n; i++) fathers[i] = -1;
        for(int i = 0; i<n; i++){
            int m; cin >> m;
            for(int j = 0; j<m; j++){
                int x; cin >> x;
                fathers[x-1] = i;
            }
        }
        init(n, fathers);
        cout << "Case " << r+1 << ":\n";
        int q; cin >> q;
        for(int a = 0; a<q; a++){
            int u, v; cin >> u >> v;
            cout << query(u-1, v-1)+1 << endl;
        }
    }
    return 0;
}

```

4.6 All Pairs Shortest Paths (Floyd Warshall)

```

const int inf = 1e9;
int main()
{
    int n, m; cin >> n >> m;
    vector<vector<int>> > D(n, vector<int>(n, inf));
    for(int i=0; i<m; i++){
        int a, b, d; cin >> a >> b >> d;
        D[a-1][b-1] = D[b-1][a-1] = d;
    }
    for(int i = 0; i<n; D[i][i] = 0);
    for(int k = 0; k<n; k++){
        for(int j = 0; j<n; j++){
            for(int i = 0; i<n; i++){
                D[i][j] = min(D[i][j], D[i][k]+D[k][j]);
            }
        }
    }

    return 0;
}

```

5 Data structures

5.1 Disjoint Sets (Union-Find)

```

vector<int> parent;

void init(int n){
    parent.resize(n);
    for(int i = 0; i<n; i++) parent[i] = i;
}

int find(int u){
    return u == parent[u] ? u : parent[u] = find(parent[u]);
}

void uni(int u, int v){
    parent[find(u)] = find(v);
}

```

5.2 Fenwick Tree

```

//Not Tested
#define LOGSZ 17

int tree[(1<<LOGSZ)+1];
int N = (1<<LOGSZ);

// add v to value at x
void set(int x, int v) {
    while(x <= N) {
        tree[x] += v;
        x += (x & -x);
    }
}

// get cumulative sum up to and including x
int get(int x) {
    int res = 0;
    while(x) {
        res += tree[x];
        x -= (x & -x);
    }
    return res;
}

// get largest value with cumulative sum less than or equal to x;
// for smallest, pass x-1 and add 1 to result
int getind(int x) {
    int idx = 0, mask = N;
    while(mask && idx < N) {
        int t = idx + mask;
        if(x >= tree[t]) {

```

```

    idx = t;
    x -= tree[t];
}
mask >>= 1;
}
return idx;
}

```

5.3 Segment Tree

```

#define inf 1e9

struct segmentTree{
    vector<int> tree;
    int size;
    segmentTree(int n){
        for(size = 2;n/=2;size+=size);
        tree.resize(size+size,inf);
    }
    void fix(int i){
        tree[i] = min(tree[i+i],tree[i+i+1]);
    }
    void update(int x,int v){
        for(tree[x+=size] = v;x/=2;fix(x));
    }
    int query(int a,int b){
        int res = inf;
        for(a += size,b+=size;a<b;a/=2,b/=2){
            if(a % 2 == 1) res = min(res,tree[a++]);
            if(b % 2 == 0) res = min(res,tree[b--]);
        }
        return (a==b? min(res,tree[a]) : res);
    }
};

```

5.4 Convex Hull Trick

```

typedef pair<ll, ll> llll;

enum line_type{
    normal,
    minus_infinity,
    infinity,
    value
};

ll gcd(ll a, ll b){
    if (a < b) swap(a, b);
    if (b == 0) return a;
    return gcd(b, a % b);
}

struct fraction{

```

```

    ll a, b;
    fraction(ll _a, ll _b = 1) : a(_a), b(_b){
        if (a == 0 && b == 0) return;
        if (b < 0) a = -a, b = -b;
        ll c = gcd(abs(a), b);
        a /= c, b /= c;
    }

    operator llll() const{
        return llll(a, b);
    }
};

inline bool operator<(const fraction& f1, const fraction& f2){
    if (llll(f1) == llll(f2)) return false;
    if (llll(f1) == llll(-1, 0) || llll(f2) == llll(1, 0)) return true;
    ;
    if (llll(f1) == llll(1, 0) || llll(f2) == llll(-1, 0)) return false;
    return ld(f1.a) * ld(f2.b) < ld(f2.a) * ld(f1.b);
}

struct line{
    ll a, b;
    line_type t = normal;
    set<line>::iterator* it = new set<line>::iterator;

    line(ll _a, ll _b) : a(_a), b(_b){}
    line(ll _x) : a(0), b(_x), t(value){}
    line(const line& other) : a(other.a), b(other.b), t(other.t){
        *it = *other.it;
    }
    line& operator=(const line& other){
        a = other.a;
        b = other.b;
        t = other.t;
        *it = *other.it;
        return *this;
    }

    ll operator()(ll x) const{
        return a * x + b;
    }

    static line left_edge(){
        line ret(0);
        ret.t = minus_infinity;
        return ret;
    }

    static line right_edge(){
        line ret(0);
        ret.t = infinity;
        return ret;
    }
}

```

```

~line(){
    delete it;
}

};

inline fraction intersection(const line& l1, const line& l2){
    if (l1.t == infinity || l2.t == infinity) return fraction(1, 0);
    if (l1.t == minus_infinity || l2.t == minus_infinity) return
        fraction(-1, 0);
    return fraction(l1.b - l2.b, l2.a - l1.a);
}

inline bool operator<(const line& l1, const line& l2){
    if (l1.t == normal && l2.t == normal) return l1.l(-l1.a, l1.b) <
        l1.l(-l2.a, l2.b);
    if (l1.t == minus_infinity || l2.t == infinity) return true;
    if (l1.t == infinity || l2.t == minus_infinity) return false;
    if (l1.t == value) return fraction(l1.b) < intersection(*prev(*l2.
        it), l2);
    return !(fraction(l2.b) < intersection(l1, *next(*l1.it)));
}

struct cht{
    set<line> s = {line::left_edge(), line::right_edge()};

    cht(){
        *(s.begin()->it) = s.begin();
        *(next(s.begin()->it) = next(s.begin()));
    }

    void insert_line(const line& l){
        set<line>::iterator it2 = s.upper_bound(l), it1 = prev(it2);
        if (it1 != s.begin() && (it1->a == l.a || !(intersection(*it1,
            l) < intersection(*it1, *it2)))) return;
        if (it2 != prev(s.end()) && l.a == it2->a) ++it2;
        while (it1 != s.begin() && !(intersection(*prev(it1), *it1) <
            intersection(*it1, l))) --it1;
        while (it2 != prev(s.end()) && !(intersection(*it2, l) <
            intersection(*it2, *next(it2)))) ++it2;
        while (next(it1) != it2) s.erase(next(it1));
        set<line>::iterator it = s.insert(it1, l);
        *(it->it) = it;
    }

    ll operator()(ll x){
        return (*prev(s.upper_bound(line(x))))(x);
    }
};

```

5.5 Linear Convex Hull Trick

```

struct fraction{
    ll a, b;
    fraction(ll _a, ll _b = 1) : a(_a), b(_b){}
};

bool operator<(const fraction& f1, const fraction& f2){
    return f1.a * f2.b < f2.a * f1.b;
}

struct line{
    ll a, b;

    line(ll _a, ll _b) : a(_a), b(_b){}

    ll operator()(ll x) const{
        return a * x + b;
    }
};

inline fraction intersection(const line& l1, const line& l2){
    return fraction(l1.b - l2.b, l2.a - l1.a);
}

struct cht{
    int pos = 0;
    vector<line> arr = {};

    cht(){}

    void insert_line_at_end(const line& l){
        if (arr.size() != 0 && !(l1.l(l.a, l.b) < l1.l(arr.back().a,
            arr.back().b))) return;
        if (arr.size() != 0 && l.a == arr.back().a) arr.pop_back();
        while (arr.size() >= 2 && !(intersection(arr[arr.size() - 2],
            arr.back()) < intersection(arr.back(), l))) arr.pop_back()
            ;
        arr.push_back(l);
    }

    ll operator()(ll x){
        pos = min(pos, int(arr.size()) - 1);
        while (pos != 0 && arr[pos - 1](x) < arr[pos](x)) --pos;
        while (pos != int(arr.size()) - 1 && arr[pos](x) >= arr[pos +
            1](x)) ++pos;
        return arr[pos](x);
    }
};

```