KUET_Crescendo Notebook (2019-20)

RUDRA_DAS | Joker_pablo | sabertooth

Contents

1 Data Structures

1.1 Ordered set PDBS

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

#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp>     // Including tree_order_statistics_node_update

using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> ordered_set;

int main() {
    ordered_set X;
    // X = {1, 2, 4, 8, 16}
    for(int i = 1; i <= 16; i *= 2)
        X.insert(i);
    cout << *X.find_by_order(0) << endl; // 1
    cout << *X.find_by_order(1) << endl; // 2
    cout << *X.find_by_order(2) << endl; // 4
    cout << *X.find_by_order(4) << endl; // 16
    cout << (X.end()==X.find_by_order(6)) << endl; // true

    cout<<X.order_of_key(-5)<<endl;  // 0
    cout<<X.order_of_key(1)<<endl;   // 0
    cout<<X.order_of_key(3)<<endl;   // 2
    cout<<X.order_of_key(4)<<endl;   // 2
    cout<<X.order_of_key(400)<<endl; // 5
}
```

1.2 Rope

```cpp
##
Here you should quickly move the block [l,r] to the beginning of the array 10^5 times
and besides the size of array is not greater than 10^5:

#include <bits/stdc++.h>
#include <ext/rope> // Slow (balanced BST)!!! do not abuse
using namespace std;
using namespace __gnu_cxx;
int main() {
    rope <int> v; //use as usual STL container
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        v.push_back(i); //initialization
    int l, r;
    for(int i = 0; i < m; ++i)
    {
        cin >> l >> r;
        --l, --r;
        rope <int> cur = v.substr(l, r - l + 1);
        v.erase(l, r - l + 1);
        v.insert(v.mutable_begin(), cur);
    }
    for(rope <int>::iterator it = v.mutable_begin(); it != v.mutable_end(); ++it)
        cout << *it << " ";
}
```

1.3 Unordered Map Pair Hash Function

```cpp
namespace std {
    template<>
    struct hash< pair<int,long long> > {
        public:
            size_t operator() (pair<int,long long> x) const {
                return x.first * 1000000009 + x.second;
            }
    };
}
```

1.4 RMQ Sparse Table 2D

```cpp
//2D RMQ
//sparse_table N*logN*M*logM
const int N = 501;
int table[N][12][N][12];
int n, m;
```

```
void create_sparse()
{
    for (int i = 0; i < n; i++)
    {
        for (int k = 1; (1 << k) <= m; k++)
            for (int j = 0; j + (1 << k) - 1 < m; j++)
                table[i][0][j][k] = max(table[i][0][j][k - 1], table[i][0][j + 1 << (k - 1)][k - 1]);
    }
    for (int pi = 1; (1 << pi) <= n; pi++)
        for (int i = 0; i < n; i++)
            for (int pj = 0; (1 << pj) <= m; pj++)
                for (int j = 0; j < m; j++)
                    table[i][pi][j][pj] = max(table[i][pi - 1][j][pj], table[i + (1 << (pi - 1))][pi - 1][j][pj]);
}
int query(int x1, int y1, int x2, int y2)
{
    int lenx = x2 - x1 + 1;
    int leny = y2 - y1 + 1;
    int kx = log(lenx);
    int ky = log(leny);
    return max(table[x1][kx][y1][ky], max(table[x1][kx][y2 + 1 - (1 << ky)][ky], max(table[x2 + 1 - (1 << kx)][kx][y1][
        ky], table[x2 + 1 - (1 << kx)][kx][y2 + 1 - (1 << ky)][ky])));
}

//Less Space Complexity
//loj square queries
const int N = 503;
unsigned int table[N][N][10];
int n, m;
void create_sparse()
{
    for (int k = 1; (1 << k) <= m; k++)
        for (int i = 0; i + (1 << k) - 1 < n; i++)
            for (int j = 0; j + (1 << k) - 1 < m; j++)
            {
                table[i][j][k] = max(table[i][j][k - 1], max(table[i][j + (1 << (k - 1))][k - 1], max(table[i + (1 << (
                    k - 1))][j][k - 1], table[i + (1 << (k - 1))][j + (1 << (k - 1))][k - 1])));
            }
}
unsigned int query(int i, int j, int S)
{
    int p = log2(S);
    assert((j + S - (1 << p)) > -1 and (j + S - (1 << p)) < n);
    assert((i + S - (1 << p)) > -1 and (i + S - (1 << p)) < n);
    return max(table[i][j][p], max(table[i][j + S - (1 << p)][p], max(table[i + S - (1 << p)][j][p], table[i + S - (1
        << p)][j + S - (1 << p)][p])));
}
```

1.5 Disjoint Set Union

```
int par[N];
int hight[N];
int find_(int x)
{
    while(x!=par[x])
    {
        par[x]=par[par[x]];
        x=par[x];
    }
    return x;
}
void UNION(int x,int y)
{
    x=find_(x);
    y=find_(y);
    if(x!=y)
    {
        if(hight[x]<hight[y])
            swap(x,y);
        par[y]=x;
        hight[x]+=hight[y];
    }
}
```

1.6 RMQ Using Sparse Table

```
/*
Dont be confused with topcoder's tutorial
use : x+2^(j-1) during initialization not x+2^(j-1)-1
Query function is ok in topcoder's tutorial
*/
const int N = 100000 + 9;
pair<int, int> tree[N][30];
int ara[N];

int myLog2(int x)
```

```
{
    return __builtin_clz(1) - __builtin_clz(x);
}
void build(int n)
{
    for (int i = 1; i <= n; i++)
    {
        tree[i][0] = MP(i, i);
    }
    int limi = myLog2(n);
    for (int j = 1; j <= limi; j++)
    {
        for (int i = 1; i + ((1 << j) - 1) <= n; i++)
        {
            if (ara[tree[i][j - 1].ff] > ara[tree[i + (1 << (j - 1))][j - 1].ff])
                tree[i][j].ff = tree[i][j - 1].ff;
            else
                tree[i][j].ff = tree[i + (1 << (j - 1))][j - 1].ff;

            if (ara[tree[i][j - 1].ss] < ara[tree[i + (1 << (j - 1))][j - 1].ss])
                tree[i][j].ss = tree[i][j - 1].ss;
            else
                tree[i][j].ss = tree[i + (1 << (j - 1))][j - 1].ss;
        }
    }
}
int query_min(int l, int r)
{
    int k = myLog2(r - l + 1);
    int val1 = ara[tree[l][k].ss];
    int val2 = ara[tree[r - (1 << k) + 1][k].ss];
    return min(val1, val2);
}
int query_max(int l, int r)
{

    int k = myLog2(r - l + 1);
    int val1 = ara[tree[l][k].ff];
    int val2 = ara[tree[r - (1 << k) + 1][k].ff];
    return max(val1, val2);
}
```

1.7 Segment Tree On Euler Path

```
int dfs_low[N];
int dfs_high[N];
ll tree_[10 * N];
int lazy[10 * N];
vector<int> edge[N];
int color[N];
int tim = 0;
void dfs(int x, int par = 0)
{
    dfs_low[x] = ++tim;
    for (int i = 0; i < edge[x].size(); i++)
    {
        if (dfs_low[edge[x][i]] == 0)
            dfs(edge[x][i], x);
    }
    dfs_high[x] = tim;
}
void push_down(int node, int st, int en)
{
    int lt = node * 2;
    int rg = node * 2 + 1;

    tree_[lt] = (1ll << lazy[node]);
    tree_[rg] = (1ll << lazy[node]);

    if (st != en)
    {
        lazy[lt] = lazy[node];
        lazy[rg] = lazy[node];
    }
    lazy[node] = 0;
}
void update1(int node, int st, int en, int l, int r, int col)
{
    if (st >= l && en <= r)
    {
        tree_[node] |= (1ll << col);
        return;
    }
    if (st > r || en < l)
        return;
    int mid = (st + en) / 2;
    int lt = node * 2;
    int rg = node * 2 + 1;
    update1(lt, st, mid, l, r, col);
    update1(rg, mid + 1, en, l, r, col);
```

```
    tree_[node] = tree_[lt] | tree_[rg];
}
ll query(int node, int st, int en, int l, int r)
{
    if (lazy[node])
        push_down(node, st, en);
    if (st > r || en < l)
        return 0ll;
    if (st >= l && en <= r)
    {
        return tree_[node];
    }
    int mid = (st + en) / 2;
    int lt = node * 2;
    int rg = node * 2 + 1;
    ll x = query(lt, st, mid, l, r);
    ll y = query(rg, mid + 1, en, l, r);
    return x | y;
}
void update2(int node, int st, int en, int l, int r, int col)
{
    if (lazy[node])
        push_down(node, st, en);
    if (st > r || en < l)
        return;
    if (st >= l && en <= r)
    {
        tree_[node] = (1ll << col);
        if (st != en)
            lazy[node] = col;
        return;
    }
    int mid = (st + en) / 2;
    int lt = node * 2;
    int rg = node * 2 + 1;
    update2(lt, st, mid, l, r, col);
    update2(rg, mid + 1, en, l, r, col);
    tree_[node] = tree_[lt] | tree_[rg];
}
int main()
{
    int n, q, a, b;
    scanf("%d%d", &n, &q);
    for (int i = 1; i <= n; i++)
        scanf("%d", &color[i]);
    for (int i = 1; i < n; i++)
    {
        scanf("%d%d", &a, &b);
        edge[a].emplace_back(b);
        edge[b].emplace_back(a);
    }
    dfs(1);
    //    for(int i=1;i<=n;i++){
    //        cerr<<i<<" "<<dfs_low[i]<<" "<<dfs_high[i]<<"\n";
    //    }
    for (int i = 1; i <= n; i++)
    {
        update1(1, 1, tim, dfs_low[i], dfs_low[i], color[i]);
        //update1(1,1,tim,dfs_high[i],dfs_high[i],color[i]);
    }
    int c;
    while (q--)
    {
        scanf("%d", &a);
        if (a == 1)
        {
            scanf("%d %d", &b, &c);
            update2(1, 1, tim, dfs_low[b], dfs_high[b], c);
        }
        else
        {
            scanf("%d", &b);
            ll x = query(1, 1, tim, dfs_low[b], dfs_high[b]);
            ll ans = total_1s(x);
            //ans=max(1ll,ans);
            printf("%I64d\n", ans);
        }
    }
}
```

1.8 Segment Tree Range Update GSS

```
//If We want to Update range L-R 1 to R-L+1
#define sum(x) x *(x + 1) / 2
void push_down(int node, ll st, ll en)
{
    if (st != en)
    {
        ll mid = (st + en) / 2, lt = node * 2, rg = node * 2 + 1;
```

```
        tree[lt].lazy = tree[rg].lazy = 1;

        tree[lt].start += tree[node].start;
        tree[lt].en += tree[node].start + (mid - st) * tree[node].inc;

        tree[rg].start += (tree[node].start + (mid - st + 1) * tree[node].inc);
        tree[rg].en += tree[node].en;

        tree[lt].inc += tree[node].inc;
        tree[rg].inc += tree[node].inc;

        tree[lt].sum += (tree[node].start * (mid - st + 1)) + tree[node].inc * sum(mid - st);

        tree[rg].sum += (tree[node].start * (en - st + 1)) + tree[node].inc * sum(en - st) - (tree[node].start * (mid -
            st + 1)) - tree[node].inc * sum(mid - st);
    }
    tree[node].en = tree[node].start = tree[node].inc = 0;
    tree[node].lazy = 0;
}
void update(int node, ll st, ll en, ll L, ll R)
{
    if (tree[node].lazy)
        push_down(node, st, en);
    if (en < L || st > R)
        return;
    if (st >= L and en <= R)
    {
        ll fx, fs;
        st == L ? fx = 1 : fx = (st - L + 1);
        en == R ? fs = R - L + 1 : fs = (R - L + 1) - (R - en);
        tree[node].sum += sum(fs) - sum(fx - 1);
        tree[node].lazy = 1;
        tree[node].start += fx;
        tree[node].en += fs;
        tree[node].inc++;
        return;
    }
    ll mid = (st + en) / 2;
    update(node * 2, st, mid, L, R);
    update(node * 2 + 1, mid + 1, en, L, R);
    tree[node].sum = tree[node * 2].sum + tree[node * 2 + 1].sum;
    tree[node].en = tree[node].start = tree[node].inc = 0;
    tree[node].lazy = 0;
}
```

1.9 LCA functions

```
//LCA
//[0->N)
const int N = 10000;
vector<pii> v[N + 2];
int level[N + 3];
int dist[N + 3];
int T[N + 2];
int sparse_table[N + 2][16];
unsigned int power[16];
int pwrsz;
void cln();
void dfs(int x, int par);
void create_sparse(int n);
int find_lca(int v1, int v2);
int find_dist(int v1, int v2);
int bin_sch(int key);
int find_k(int v1, int k);
int find_kth_par(int v1, int v2, int K);
int main()
{
    for (int i = 0; (1 << i) <= N; i++)
    {
        power[i] = (1 << i);
        pwrsz++;
    }
    fast;
    int test;
    cin >> test;
    while (test--)
    {
        cln();
        int n;
        cin >> n;
        int a, b, c;
        for (int i = 1; i < n; i++)
        {
            cin >> a >> b >> c;
            a--;
            b--;
            v[a].pb(make_pair(b, c));
            v[b].pb(make_pair(a, c));
        }
        dfs(0, 0);
```

```
        create_sparse(n);
        string s;
        while (cin >> s)
        {
            if (s[0] == 'D' and s[1] == 'O')
                break;
            cin >> a >> b;
            a--;
            b--;
            if (s[0] == 'D')
                cout << find_dist(a, b) << endl;
            else
            {
                cin >> c;
                c--;
                cout << find_kth_par(a, b, c) + 1 << endl;
            }
        }
    }
}
void cln()
{
    for (int i = 0; i <= N; i++)
    {
        v[i].clear();
        level[i] = 0;
        T[i] = 0;
        dist[i] = 0;
    }
}
void dfs(int x, int par)
{
    T[x] = par;
    int u, dis;
    for (int i = 0; i < (int)v[x].size(); i++)
    {
        u = v[x][i].ff;
        dis = v[x][i].ss;
        if (par == u)
            continue;
        level[u] = level[x] + 1;
        dist[u] = dist[x] + dis;
        dfs(u, x);
    }
}
void create_sparse(int n)
{
    memset(sparse_table, -1, sizeof sparse_table);
    for (int i = 0; i < n; i++)
        sparse_table[i][0] = T[i];
    for (int j = 1; (1 << j) < n; j++)
        for (int i = 0; i < n; i++)
            if (sparse_table[i][j - 1] != -1)
                sparse_table[i][j] = sparse_table[sparse_table[i][j - 1]][j - 1];
}
int find_lca(int v1, int v2)
{
    if (level[v1] < level[v2]) //Bigger level in V1
        swap(v1, v2);
    int lg = 1;
    int nxt = lg + 1;
    while ((1 << nxt) <= level[v1]) //highest depth for 2^i
    {
        lg++;
        nxt = lg + 1;
    }
    for (int i = lg; i > -1; i--)
        if (level[v1] - (1 << i) >= level[v2])
            v1 = sparse_table[v1][i];
    if (v1 == v2)
        return v1;
    for (int i = lg; i >= 0; i--)
        if (sparse_table[v1][i] != -1 and sparse_table[v1][i] != sparse_table[v2][i])
            v1 = sparse_table[v1][i], v2 = sparse_table[v2][i];
    return T[v1];
}
int find_dist(int v1, int v2)
{
    int x = find_lca(v1, v2);
    return dist[v1] + dist[v2] - 2 * dist[x];
}
int bin_sch(int key)
{
    int ans = 0, lo = 0, hi = pwrsz - 1;
    int mid;
    while (hi >= lo)
    {
        mid = (hi + lo) / 2;
        if (power[mid] <= key)
            ans = mid, lo = mid + 1;
        else
            hi = mid - 1;
    }
```

```
    return ans;
}
int find_k(int v1, int k)
{
    int ans = v1, j;
    while (k > 0)
    {
        j = bin_sch(k);
        k -= power[j];
        ans = sparse_table[ans][j];
    }
    return ans;
}
int find_kth_par(int v1, int v2, int K)
{
    int x = find_lca(v1, v2);
    int X = level[v1] - level[x];
    int Y = level[v2] - level[x];
    if (K <= X)
        return find_k(v1, K);
    else
        return find_k(v2, Y - K + X);
}
```

1.10 Heavy Light Decomposition

```
// Tested:
// - http://codeforces.com/gym/100739/problem/G
// Notes:
// - Index from 1
// - dfn: vertex id --> position
// --> for point query, only need to visit(dfn[u])
// --> for range query, use query(u, v)
// Author: zimpha
#include <bits/stdc++.h>
using namespace std;
const int MN = 100111;
vector<int> G[MN];
int sz[MN], dep[MN], fa[MN];
int dfn[MN], top[MN];
int n, id;

// Operations on segment tree
int querySeg(int i, int l, int r, int u, int v) {return 0;}

void dfs1(int u, int f = 0) {
    sz[u] = 1; fa[u] = f;
    for (auto &v: G[u]) if (v != f) {
        dep[v] = dep[u] + 1;
        dfs1(v, u); sz[u] += sz[v];
    }
}

void dfs2(int u, int chain, int f = 0) {
    int son(-1); dfn[u] = ++id; top[u] = chain;
    for (auto &v: G[u]) if (v != f) {
        if (son == -1 || sz[son] < sz[v]) son = v;
    }
    if (~son) dfs2(son, chain, u);
    for (auto &v: G[u]) if (v != f && v != son) {
        dfs2(v, v, u);
    }
}

int query(int u, int v) {
    int res = 0;
    int fu = top[u], fv = top[v];
    while (fu != fv) {
        if (dep[fu] < dep[fv]) swap(u, v), swap(fu, fv);
        res += querySeg(1, 1, n, dfn[fu], dfn[u]);
        u = fa[fu], fu = top[u];
    }
    if (dep[u] > dep[v]) swap(u, v);
    res += querySeg(1, 1, n, dfn[u], dfn[v]);
    return res;
}

int main() {
    dfs1(1);
    id = 0;
    dfs2(1, 1);
}
```

1.11 Heavy Light Decomposition Adamant Implementation

```
// Usage:
```

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

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

1.12 MoAlgorithms Usage

```
Mo's algorithm:
- Sort by (l / SQRTN, r)
--> Extend the segment in O(|l - l'| + |r - r'|)
On tree:
- Standard method of linearizing does not work, because nodes can be O(N) apart
- Slightly modify the standard dfs traversal, by adding nodes at even depth to our traversal as we go down, and nodes
    at odd depth to our traversal as we go up, kind of "hopping" down and up the tree. This ensures that nodes right
    next to each other in the traversal are at most 3 nodes apart in the actual tree
```

2 Dynamic Programming

2.1 Convex Hull

```
// Original Recurrence:
//   dp[i] = min( dp[j] + b[j]*a[i] )  for j < i
// Condition:
//   b[j] >= b[j+1]
//   a[i] <= a[i+1]
// To solve:
// Hull hull;
// FOR(i,1,n) {
//   dp[i] = hull.get(a[i]);
//   hull.add(b[i], dp[i]);
// }

const int MAXN = 100100;

struct Hull {
    long long a[MAXN], b[MAXN];
    double x[MAXN];
    int head, tail;

    Hull(): head(1), tail(0) {}

    long long get(long long xQuery) {
        if (head > tail) return 0;
        while (head < tail && x[head + 1] <= xQuery) head++;
        x[head] = xQuery;
        return a[head] * xQuery + b[head];
    }

    void add(long long aNew, long long bNew) {
        double xNew = -1e18;
        while (head <= tail) {
            if (aNew == a[tail]) return;
            xNew = 1.0 * (b[tail] - bNew) / (aNew - a[tail]);
            if (head == tail || xNew >= x[tail]) break;
            tail--;
        }
        a[++tail] = aNew;
        b[tail] = bNew;
        x[tail] = xNew;
    }
};
```

2.2 Convex Hull 2

```
// Tested: http://codeforces.com/contest/678/standings/friends/true
// Add lines a*x + b, must be in increasing order of a
// Get y = max(a*x + b)
struct Hull {
    vector<double> x;
    vector<int> a;
    vector<int> b;

    void init() {
        x.clear();
        a.clear();
        b.clear();
    }

    void remove() {
        a.pop_back();
        b.pop_back();
        x.pop_back();
    }

    void insert(Line l) {
        if (a.empty()) {
            x.push_back(-INF);
            a.push_back(l.a);
            b.push_back(l.b);
        }
        else {
            double xNew = -INF;
            while (!a.empty()) {
                if (a.back() == l.a) {
                    b.back() = max(b.back(), l.b);
                    return;
                }
                assert(l.a > a.back());

                xNew = 1.0 * (b.back() - l.b) / (l.a - a.back());
                if (xNew < x.back()) {
                    remove();
                }
                else break;
            }

            a.push_back(l.a);
            b.push_back(l.b);
            x.push_back(xNew);
        }
    }

    int get(int x0) {
        if (a.empty()) {
            return -INF;
        }
        int i = upper_bound(x.begin(), x.end(), x0) - x.begin() - 1;
        return a[i] * x0 + b[i];
    }
};
```

2.3 Divide And Conquer

```
// http://codeforces.com/blog/entry/8219
// Divide and conquer optimization:
// Original Recurrence
//   dp[i][j] = min(dp[i-1][k] + C[k][j]) for k < j
// Sufficient condition:
//   A[i][j] <= A[i][j+1]
//   where A[i][j] = smallest k that gives optimal answer
// How to use:
//   // compute i-th row of dp from L to R. optL <= A[i][L] <= A[i][R] <= optR
//   compute(i, L, R, optL, optR)
//       1. special case L == R
//       2. let M = (L + R) / 2. Calculate dp[i][M] and opt[i][M] using O(optR - optL + 1)
//       3. compute(i, L, M-1, optL, opt[i][M])
//       4. compute(i, M+1, R, opt[i][M], optR)

// Example: http://codeforces.com/contest/321/problem/E
#include "../template.h"

const int MN = 4011;
const int inf = 1000111000;
int n, k, cost[MN][MN], dp[811][MN];

inline int getCost(int i, int j) {
    return cost[j][j] - cost[j][i-1] - cost[i-1][j] + cost[i-1][i-1];
}

void compute(int i, int L, int R, int optL, int optR) {
    if (L > R) return ;
```

```
    int mid = (L + R) >> 1, savek = optL;
    dp[i][mid] = inf;
    FOR(k,optL,min(mid-1, optR)) {
        int cur = dp[i-1][k] + getCost(k+1, mid);
        if (cur < dp[i][mid]) {
            dp[i][mid] = cur;
            savek = k;
        }
    }
    compute(i, L, mid-1, optL, savek);
    compute(i, mid+1, R, savek, optR);
}

void solve() {
    cin >> n >> k;
    FOR(i,1,n) FOR(j,1,n) {
        cin >> cost[i][j];
        cost[i][j] = cost[i-1][j] + cost[i][j-1] - cost[i-1][j-1] + cost[i][j];
    }
    dp[0][0] = 0;
    FOR(i,1,n) dp[0][i] = inf;

    FOR(i,1,k) {
        compute(i, 1, n, 0, n);
    }
    cout << dp[k][n] / 2 << endl;
}
```

2.4 Dynamic Hull

```
// source: https://github.com/niklasb/contest-algos/blob/master/convex_hull/dynamic.cpp
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

2.5 Knuth Optimizations

```
// http://codeforces.com/blog/entry/8219
// Original Recurrence:
//   dp[i][j] = min(dp[i][k] + dp[k][j]) + C[i][j]   for k = i+1..j-1
// Necessary & Sufficient Conditions:
//   A[i][j-1] <= A[i][j] <= A[i+1][j]
//   with A[i][j] = smallest k that gives optimal answer
// Also applicable if the following conditions are met:
//   1. C[a][c] + C[b][d] <= C[a][d] + C[b][c] (quadrangle inequality)
//   2. C[b][c] <= C[a][d]                     (monotonicity)
//   for all a <= b <= c <= d
// To use:
//   Calculate dp[i][i] and A[i][i]
//
//   FOR(len = 1..n-1)
//     FOR(i = 1..n-len) {
```

```
//       j = i + len
//       FOR(k = A[i][j-1]..A[i+1][j])
//         update(dp[i][j])
//     }

// OPTCUT
#include "../template.h"

const int MN = 2011;
int a[MN], dp[MN][MN], C[MN][MN], A[MN][MN];
int n;

void solve() {
    cin >> n; FOR(i,1,n) { cin >> a[i]; a[i] += a[i-1]; }
    FOR(i,1,n) FOR(j,i,n) C[i][j] = a[j] - a[i-1];

    FOR(i,1,n) dp[i][i] = 0, A[i][i] = i;

    FOR(len,1,n-1)
        FOR(i,1,n-len) {
            int j = i + len;
            dp[i][j] = 2000111000;
            FOR(k,A[i][j-1],A[i+1][j]) {
                int cur = dp[i][k-1] + dp[k][j] + C[i][j];
                if (cur < dp[i][j]) {
                    dp[i][j] = cur;
                    A[i][j] = k;
                }
            }
        }
    cout << dp[1][n] << endl;
}
```

2.6 Longest Increasing Subseqence

```
// Source: http://codeforces.com/blog/entry/13225
// Non-strict.
multiset<int> s;
vector<int> a;

for (int i = 1; i <= n; i++) {
    s.insert(a[i]);
    auto it = s.upper_bound(a[i]);

    if (it != s.end())
        s.erase(it);
}
cout << s.size() << endl;

// Strict.
multiset<int> s;
for (int i = 1; i <= n; i++) {
    s.insert(a[i]);
    it = s.lower_bound(a[i]);
    it++;

    if (it != s.end())
        s.erase(it);
}

// Trace
vector<int> b(n+1, 0);
int answer = 0;
for (int i = 1; i <= n; i++) {
    f[i] = lower_bound(b+1, b+answer+1, a[i]) - b;
    answer = max(answer, f[i]);
    b[f[i]] = a[i];
}

int require = answer;
vector<int> T;
for (int i = n; i >= 1; i--) {
    if (f[i] == require) {
        T.push_back(a[i]);
        require--;
    }
}
reverse(T.begin(), T.end());
```

3 Geometry

3.1 Geometry template 2d

```
#include "template.h"
```

```
// Experimenting
// Tested:
// - http://codeforces.com/gym/100803 - H
// - http://codeforces.com/gym/100834 - E
// - http://codeforces.com/gym/100506 - H
// Tested vs Geometry v1:
// - 100506H: slower (4.196 vs 3.666)

const double EPS = 1e-6;
const double INF = 1e9;

int cmp(double x, double y) {
    if (fabs(x - y) < EPS) return 0;
    if (x < y) return -1;
    return 1;
}

// ------------------------ BASIC TYPE
struct D {
    double x;

        D() {}
    D(double x) : x(x) {}

    D operator + (const D& a) const { return D(x+a.x); }
    D operator - (const D& a) const { return D(x-a.x); }
    D operator * (const D& a) const { return D(x*a.x); }
    D operator / (const D& a) const { return D(x/a.x); }

    D operator - () const { return D(-x); }

    D& operator += (const D& a) { return *this = *this + a; }
    D& operator -= (const D& a) { return *this = *this - a; }
    D& operator *= (const D& a) { return *this = *this * a; }
    D& operator /= (const D& a) { return *this = *this / a; }

    bool operator == (const D& a) const { return cmp(x, a.x) == 0; }
    bool operator <= (const D& a) const { return cmp(x, a.x) <= 0; }
    bool operator >= (const D& a) const { return cmp(x, a.x) >= 0; }
    bool operator <  (const D& a) const { return cmp(x, a.x) <  0; }
    bool operator >  (const D& a) const { return cmp(x, a.x) >  0; }
    bool operator != (const D& a) const { return cmp(x, a.x) != 0; }

    int sign() {
        int t = cmp(x, 0);
        if (t == 0) return 0;
        if (t < 0) return -1;
        return 1;
    }

    friend istream& operator >> (istream& cin, D& x) {
        cin >> x.x;
        return cin;
    }
    friend ostream& operator << (ostream& cout, D& x) {
        cout << x.x;
        return cout;
    }
} O(0.0), PI(acos((double) -1.0));

int cmp(const D& a, const D& b) {
    return cmp(a.x, b.x);
}

D sqrt(D x) { assert(x >= 0); return D(sqrt(x.x)); }
D abs(D x) { if (x < 0) return -x; else return x; }
D fabs(D x) { if (x < 0) return -x; else return x; }
D sin(D x) { return sin(x.x); }
D cos(D x) { return cos(x.x); }
D tan(D x) { return tan(x.x); }
D asin(D x) { assert(D(-1) <= x && x <= D(1)); return asin(x.x); }
D acos(D x) { assert(D(-1) <= x && x <= D(1)); return acos(x.x); }
D atan(D x) { return atan(x.x); }
D atan2(D a, D b) { return atan2(a.x, b.x); }

struct Point {
    D x, y;

    Point() {}
    Point(double x, double y) : x(x), y(y) {}
    Point(D x, D y) : x(x), y(y) {}

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

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

    Point operator + (const Point& a) const { return Point(x+a.x, y+a.y); }
    Point operator - (const Point& a) const { return Point(x-a.x, y-a.y); }
    Point operator * (const D& k)     const { return Point(x*k, y*k); }
    Point operator / (const D& k)     const { assert(k != D(0)); return Point(x/k, y/k); }

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

    friend istream& operator >> (istream& cin, Point& a) {
```

```
        cin >> a.x >> a.y;
        return cin;
    }
    friend ostream& operator << (ostream& cout, Point& a) {
        cout << a.x << ' ' << a.y;
        return cout;
    }

    D norm() { return x*x + y*y; }
    D len() { return hypot(x.x, y.x); }

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

    Point normalize(D l) {
        return Point(x, y) * l / len();
    }
};
D angle(Point a, Point o, Point b) { // min of directed angle AOB & BOA
    a = a - o; b = b - o;
    return acos((a * b) / sqrt(a.norm() * b.norm()));
}
D directed_angle(Point a, Point o, Point b) { // angle AOB, in range [0, 2*PI)
    D t = -atan2(a.y - o.y, a.x - o.x)
          + atan2(b.y - o.y, b.x - o.x);
    while (t < 0) t += D(2)*PI;
    return t;
}

int ccw(const Point& a, const Point& b, const Point& c) {
    return ((b - a) % (c - a)).sign();
}

struct Line {
    D a, b, c;
    Point A, B;

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

    Line(Point A, Point B) : A(A), B(B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = D(0) - (a * A.x + b * A.y);
    }
    D f(const Point &p) {
        return a*p.x + b*p.y + c;
    }

    D dist(Point p) {
        return fabs(a*p.x + b*p.y + c) / sqrt(a*a + b*b);
    }
};

struct Circle : Point {
    D r;

    Circle() {}
    Circle(Point a, D r) : Point(a), r(r) {}

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

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

// Distance from p to segment ab (closest Point --> c)
D distToLineSegment(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    D u = (ap * ab) / ab.norm();
    if (u < 0) {
        c = Point(a.x, a.y);
        return (p - a).len();
    }
    if (u > D(1.0)) {
        c = Point(b.x, b.y);
        return (p - b).len();
    }
    return distToLine(p, a, b, c);
}
```

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

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

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

void closestPoint(Line l, Point p, Point& ans) {
        if (l.b <= 0) {
                ans.x = -(l.c) / l.a; ans.y = p.y;
        }
        else if (l.a <= 0) {
                ans.x = p.x; ans.y = -(l.c) / l.b;
        }
        else {
                Line perp(l.b, -l.a, -(l.b*p.x - l.a*p.y));
                areIntersect(l, perp, ans);
        }
}
void reflectionPoint(Line l, Point p, Point &ans) {
    Point b;
    closestPoint(l, p, b);
    ans = p + (b - p) * 2;
}

D segment_union(vector< pair<D, D> > segs) {
    int n = SZ(segs);
    vector< pair<D, bool> > x(n*2);
    REP(i,n) {
        x[i*2] = make_pair(segs[i].first, false);
        x[i*2+1] = make_pair(segs[i].second, true);
    }
    sort(x.begin(), x.end());

    D res = 0.0;
    int c = 0;
    REP(i,n*2) {
        if (c && i) res += x[i].first - x[i-1].first;
        if (x[i].second) ++c;
        else --c;
    }
    return res;
}

// ------------------------ Circle operations
bool areIntersect(Circle u, Circle v) {
    if (cmp((u - v).len(), u.r + v.r) > 0) return false;
    if (cmp((u - v).len() + v.r, u.r) < 0) return false;
    if (cmp((u - v).len() + u.r, v.r) < 0) return false;
    return true;
}

// helper functions for commonCircleArea
D cir_area_solve(D a, D b, D c) {
    return acos((a*a + b*b - c*c) / 2 / a / b);
}
D cir_area_cut(D a, D r) {
    D s1 = a * r * r / 2;
    D s2 = sin(a) * r * r / 2;
    return s1 - s2;
}
// Tested: http://codeforces.com/contest/600/problem/D
D commonCircleArea(Circle c1, Circle c2) { //return the common area of two circle
    if (c1.r < c2.r) swap(c1, c2);
    D d = (c1 - c2).len();
    if (d + c2.r <= c1.r) return c2.r*c2.r*PI;
    if (d >= c1.r + c2.r) return 0;
    D a1 = cir_area_solve(d, c1.r, c2.r);
    D a2 = cir_area_solve(d, c2.r, c1.r);
    return cir_area_cut(a1*2, c1.r) + cir_area_cut(a2*2, c2.r);
}

vector<Point> circleIntersect(Circle u, Circle v) {
    vector<Point> res;
    if (!areIntersect(u, v)) return res;
    D d = (u - v).len();
    D alpha = acos((u.r * u.r + d*d - v.r * v.r) / 2.0 / u.r / d);

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

```
// Circle & line intersection
vector<Point> intersection(Line l, Circle cir) {
    D r = cir.r, a = l.a, b = l.b, c = l.c + l.a*cir.x + l.b*cir.y;
    vector<Point> res;

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

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

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

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

// ------------------------ Polygon
typedef vector< Point > Polygon;

D area2(Point a, Point b, Point c) { return a%b + b%c + c%a; }
bool between(const Point &a, const Point &b, const Point &c) {
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
void ConvexHull(vector<Point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<Point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

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

```
#endif
}
D signed_area(Polygon p) {
    D area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}
D area(const Polygon &p) {
    return fabs(signed_area(p));
}

D segment_union(vector< pair<D, D> > segs) {
    int n = SZ(segs);
    vector< pair<D, bool> > x(n*2);
    REP(i,n) {
        x[i*2] = make_pair(segs[i].first, false);
        x[i*2+1] = make_pair(segs[i].second, true);
    }
    sort(x.begin(), x.end());

    D res = 0.0;
    int c = 0;
    REP(i,n*2) {
        if (c && i) res += x[i].first - x[i-1].first;
        if (x[i].second) ++c;
        else --c;
    }
    return res;
}
// Missing: polygon.h
```

3.2 Geometry Template Rezaul bhai

```
#include <algorithm>
#include <cassert>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

const double pi = acos(-1.0);
const double eps = 1e-8;

typedef double T;
struct pt
{
    T x, y;
    pt() {}
    pt(T _x, T _y) : x(_x), y(_y) {}
    pt operator+(const pt &p) const
    {
        return pt(x + p.x, y + p.y);
    }
    pt operator-(const pt &p) const
    {
        return pt(x - p.x, y - p.y);
    }
    pt operator*(const T &d) const
    {
        return pt(x * d, y * d);
    }
    pt operator/(const T &d) const
    {
        return pt(x / d, y / d);
    }
    bool operator==(const pt &p) const
    {
        return (x == p.x and y == p.y);
    }
    bool operator!=(const pt &p) const
    {
        return !(x == p.x and y == p.y);
    }
    bool operator<(const pt &p) const
    {
        if (x != p.x)
            return x < p.x;
        return y < p.y;
    }
};

T sq(pt p)
{
    return p.x * p.x + p.y * p.y;
}
```

```
double abs(pt p)
{
    return sqrt(sq(p));
}

pt translate(pt v, pt p)
{
    return p + v;
}

pt scale(pt c, double factor, pt p)
{
    return c + (p - c) * factor;
}

pt rot(pt p, double a)
{
    return pt(p.x * cos(a) - p.y * sin(a), p.x * sin(a) + p.y * cos(a));
}

pt perp(pt p)
{
    return pt(-p.y, p.x);
}

T dot(pt v, pt w)
{
    return v.x * w.x + v.y * w.y;
}

bool isPerp(pt v, pt w)
{
    return dot(v, w) == 0;
}

double smallAngle(pt v, pt w)
{
    double cosTheta = dot(v, w) / abs(v) / abs(w);
    if (cosTheta < -1)
        cosTheta = -1;
    if (cosTheta > 1)
        cosTheta = 1;
    return acos(cosTheta);
}

T cross(pt v, pt w)
{
    return v.x * w.y - v.y * w.x;
}

T orient(pt a, pt b, pt c)
{
    return cross(b - a, c - a);
}

bool inAngle(pt a, pt b, pt c, pt x)
{
    assert(orient(a, b, c) != 0);
    if (orient(a, b, c) < 0)
        swap(b, c);
    return orient(a, b, x) >= 0 and orient(a, c, x) <= 0;
}

//Line
struct line
{
    pt v;
    T c;
    line() {}
    //From points P and Q
    line(pt p, pt q)
    {
        v = (q - p);
        c = cross(v, p);
    }
    //From equation ax + by = c
    line(T a, T b, T c)
    {
        v = pt(b, -a);
        c = c;
    }
    //From direction vector v and offset c
    line(pt v, T c)
    {
        v = v;
        c = c;
    }

    //These work with T = int / double
    T side(pt p);
    double dist(pt p);
    double sqDist(pt p);
    line perpThrough(pt p);
    bool cmpProj(pt p, pt q);
    line translate(pt t);

    //These require T = double
```

```
    line shiftLeft(double dist);
    pt proj(pt p);
    pt refl(pt p);
};

T line ::side(pt p)
{
    return cross(v, p) - c;
}

double line ::dist(pt p)
{
    return abs(side(p)) / abs(v);
}

double line ::sqDist(pt p)
{
    return side(p) * side(p) / (double)sq(v);
}

line line ::perpThrough(pt p)
{
    return line(p, p + perp(v));
}

bool line ::cmpProj(pt p, pt q)
{
    return dot(v, p) < dot(v, q);
}

line line ::translate(pt t)
{
    return line(v, c + cross(v, t));
}

line line ::shiftLeft(double dist)
{
    return line(v, c + dist * abs(v));
}

bool areParallel(line l1, line l2)
{
    return (l1.v.x * l2.v.y == l1.v.y * l2.v.x);
}

bool areSame(line l1, line l2)
{
    return areParallel(l1, l2) and (l1.v.x * l2.c == l2.v.x * l1.c) and (l1.v.y * l2.c == l2.v.y * l1.c);
}

bool inter(line l1, line l2, pt &out)
{
    T d = cross(l1.v, l2.v);
    if (d == 0)
        return false;
    out = (l2.v * l1.c - l1.v * l2.c) / d;
    return true;
}

pt line ::proj(pt p)
{
    return p - perp(v) * side(p) / sq(v);
}

pt line ::refl(pt p)
{
    return p - perp(v) * 2 * side(p) / sq(v);
}

line intBisector(line l1, line l2, bool interior)
{
    assert(cross(l1.v, l2.v) != 0);
    double sign = interior ? 1 : -1;
    return line(l2.v / abs(l2.v) + l1.v * sign / abs(l1.v),
                l2.c / abs(l2.v) + l1.c * sign / abs(l1.v));
}

//Segment
bool inDisk(pt a, pt b, pt p)
{
    return dot(a - p, b - p) <= 0;
}

bool onSegment(pt a, pt b, pt p)
{
    return orient(a, b, p) == 0 and inDisk(a, b, p);
}

bool properInter(pt a, pt b, pt c, pt d, pt &i)
{
    double oa = orient(c, d, a),
           ob = orient(c, d, b),
           oc = orient(a, b, c),
           od = orient(a, b, d);

    //Proper intersection exists iff opposite signs
    if (oa * ob < 0 and oc * od < 0)
```

```
        {
            i = (a * ob - b * oa) / (ob - oa);
            return true;
        }
        return false;
    }

    bool inters(pt a, pt b, pt c, pt d)
    {
        pt out;
        if (properInter(a, b, c, d, out))
            return true;
        if (onSegment(c, d, a))
            return true;
        if (onSegment(c, d, b))
            return true;
        if (onSegment(a, b, c))
            return true;
        if (onSegment(a, b, d))
            return true;
        return false;
    }

    double segPoint(pt a, pt b, pt p)
    {
        if (a != b)
        {
            line l(a, b);
            if (l.cmpProj(a, p) and l.cmpProj(p, b))
                return l.dist(p);
        }
        return min(abs(p - a), abs(p - b));
    }

    double segSeg(pt a, pt b, pt c, pt d)
    {
        pt dummy;
        if (properInter(a, b, c, d, dummy))
            return 0;
        return min(min(min(segPoint(a, b, c), segPoint(a, b, d)), segPoint(c, d, a)), segPoint(c, d, b));
    }

    //int latticePoints (pt a, pt b){
    //    //requires int representation
    //    return __gcd (abs (a.x - b.x), abs (a.y - b.y)) + 1;
    //}

    bool isConvex(vector<pt> &p)
    {
        bool hasPos = false, hasNeg = false;
        for (int i = 0, n = p.size(); i < n; i++)
        {
            int o = orient(p[i], p[(i + 1) % n], p[(i + 2) % n]);
            if (o > 0)
                hasPos = true;
            if (o < 0)
                hasNeg = true;
        }
        return !(hasPos and hasNeg);
    }

    double areaTriangle(pt a, pt b, pt c)
    {
        return abs(cross(b - a, c - a)) / 2.0;
    }

    double areaPolygon(const vector<pt> &p)
    {
        double area = 0.0;
        for (int i = 0, n = p.size(); i < n; i++)
            area += cross(p[i], p[(i + 1) % n]);
        return fabs(area) / 2.0;
    }

    bool pointInPolygon(const vector<pt> &p, pt q)
    {
        bool c = false;
        for (int i = 0, n = p.size(); i < n; i++)
        {
            int j = (i + 1) % p.size();
            if ((p[i].y <= q.y and q.y < p[j].y or p[j].y <= q.y and q.y < p[i].y) and
                q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
                c = !c;
        }
        return c;
    }

    pt centroidPolygon(const vector<pt> &p)
    {
        pt c(0, 0);
        double scale = 6.0 * areaPolygon(p);
        //    if (scale < eps) return c;
        for (int i = 0, n = p.size(); i < n; i++)
        {
            int j = (i + 1) % n;
            c = c + (p[i] + p[j]) * cross(p[i], p[j]);
        }
```

```
    return c / scale;
}

//Circle
pt circumCenter(pt a, pt b, pt c)
{
    b = b - a;
    c = c - a;
    assert(cross(b, c) != 0);
    return a + perp(b * sq(c) - c * sq(b)) / cross(b, c) / 2;
}

bool circle2PtsRad(pt p1, pt p2, double r, pt &c)
{
    double d2 = sq(p1 - p2);
    double det = r * r / d2 - 0.25;
    if (det < 0.0)
        return false;
    double h = sqrt(det);
    c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
    c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
    return true;
}

int circleLine(pt c, double r, line l, pair<pt, pt> &out)
{
    double h2 = r * r - l.sqDist(c);
    if (h2 < 0)
        return 0; // the line doesn't touch the circle;
    pt p = l.proj(c);
    pt h = l.v * sqrt(h2) / abs(l.v);
    out = make_pair(p - h, p + h);
    return 1 + (h2 > 0);
}

int circleCircle(pt c1, double r1, pt c2, double r2, pair<pt, pt> &out)
{
    pt d = c2 - c1;
    double d2 = sq(d);
    if (d2 == 0)
    { //concentric circles
        assert(r1 != r2);
        return 0;
    }
    double pd = (d2 + r1 * r1 - r2 * r2) / 2;
    double h2 = r1 * r1 - pd * pd / d2; // h ^ 2
    if (h2 < 0)
        return 0;
    pt p = c1 + d * pd / d2, h = perp(d) * sqrt(h2 / d2);
    out = make_pair(p - h, p + h);
    return 1 + h2 > 0;
}

int tangents(pt c1, double r1, pt c2, double r2, bool inner, vector<pair<pt, pt>> &out)
{
    if (inner)
        r2 = -r2;
    pt d = c2 - c1;
    double dr = r1 - r2, d2 = sq(d), h2 = d2 - dr * dr;
    if (d2 == 0 or h2 < 0)
    {
        assert(h2 != 0);
        return 0;
    }
    for (int sign : {-1, 1})
    {
        pt v = pt(d * dr + perp(d) * sqrt(h2) * sign) / d2;
        out.push_back(make_pair(c1 + v * r1, c2 + v * r2));
    }
    return 1 + (h2 > 0);
}

//Convex Hull - Monotone Chain
pt H[100000 + 5];
int monotoneChain(vector<pt> &points)
{
    sort(points.begin(), points.end());
    int st = 0;
    for (int i = 0, sz = points.size(); i < sz; i++)
    {
        while (st >= 2 and orient(H[st - 2], H[st - 1], points[i]) < 0)
            st--;
        H[st++] = points[i];
    }
    int taken = st - 1;
    for (int i = points.size() - 2; i >= 0; i--)
    {
        while (st >= taken + 2 and orient(H[st - 2], H[st - 1], points[i]) < 0)
            st--;
        H[st++] = points[i];
    }
    return st
}
```

3.3 Checks if rectangle of sides x,y fits inside one of sides X,Y

```
// Checks if rectangle of sides x,y fits inside one of sides X,Y
// Not tested with doubles but should work fine :)
// Code as written rejects rectangles that just touch.
bool rect_in_rect(int X, int Y, int x, int y) {
    if (Y > X) swap(Y, X);
    if (y > x) swap(y, x);
    double diagonal = sqrt(double(X)*X + double(Y)*Y);
    if (x < X && y < Y) return true;
    else if (y >= Y || x >= diagonal) return false;
    else {
        double w, theta, tMin = PI/4, tMax = PI/2;
        while (tMax - tMin > EPS) {
            theta = (tMax + tMin)/2.0;
            w = (Y-x*cos(theta))/sin(theta);
            if (w < 0 || x * sin(theta) + w * cos(theta) < X) tMin = theta;
            else tMax = theta;
        }
        return (w > y);
    }
}
```

3.4 Smallest circle enclosing given points

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

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

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

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

        return c;
    }

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

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

3.5 Soddy Circles

```
// Tested:
// - https://www.e-olymp.com/en/problems/269 (inner)

Given 3 mutually tangent circles. Find inner circle (touching all 3) and outer circle (touching all 3).
The radius is given by:
k4 = |k1 + k2 + k3 +- 2*sqrt(k1*k2 + k2*k3 + k3*k1)|
where ki = 1/ri
Minus --> Outer
Plus --> Inner

Special cases:
- If 1 circle --> line, change ki to 0 --> k4 = k1 + k2 +- 2*sqrt(k1*k2)
```

3.6 Spherical Co Ordinates

```
struct Point3 {
    double x, y, z;

    Point3(Spherical P) {
        x = P.r * cos(P.theta) * sin(P.phi);
        y = P.r * sin(P.theta) * sin(P.phi);
        z = P.r * cos(P.phi);
    }
};

// http://mathworld.wolfram.com/images/eps-gif/SphericalCoordinates_1201.gif
struct Spherical {
    double r,
        theta, // 0 <= theta < 2*PI
        phi;   // 0 <= phi <= PI

    Spherical(Point3 P) {
        r = sqrt(P.x*P.x + P.y*P.y + P.z*P.z);
        theta = atan(P.y / P.x); if (theta < 0) theta += 2 * PI;
        phi = acos(P.z / r);
    }
};
```

3.7 Voronoi diagrams and Convex hull

```
// Source: http://web.mit.edu/~ecprice/acm/acm08/notebook.html#file7
#define MAXN 1024
#define INF 1000000

//Voronoi diagrams: O(N^2*LogN)
//Convex hull: O(N*LogN)
typedef struct {
  int id;
  double x;
  double y;
  double ang;
} chp;

int n;
double x[MAXN], y[MAXN]; // Input points
chp inv[2*MAXN]; // Points after inversion (to be given to Convex Hull)
int vors;
int vor[MAXN]; // Set of points in convex hull;
               //starts at lefmost; last same as first!!
PT ans[MAXN][2];

int chpcmp(const void *aa, const void *bb) {
  double a = ((chp *)aa)->ang;
  double b = ((chp *)bb)->ang;
  if (a<b) return -1;
  else if (a>b) return 1;
  else return 0; // Might be better to include a
                 // tie-breaker on distance, instead of the cheap hack below
}

int orient(chp *a, chp *b, chp *c) {
  double s = a->x*(b->y-c->y) + b->x*(c->y-a->y) + c->x*(a->y-b->y);
  if (s>0) return 1;
  else if (s<0) return -1;
  else if (a->ang==b->ang && a->ang==c->ang) return -1; // Cheap hack
           //for points with same angles
  else return 0;
}

//the pt argument must have the points with precomputed angles (atan2()'s)
//with respect to a point on the inside (e.g. the center of mass)
int convexHull(int n, chp *pt, int *ans) {
  int i, j, st, anses=0;

  qsort(pt, n, sizeof(chp), chpcmp);
  for (i=0; i<n; i++) pt[n+i] = pt[i];
  st = 0;
  for (i=1; i<n; i++) { // Pick leftmost (bottommost)
                        //point to make sure it's on the convex hull
    if (pt[i].x<pt[st].x || (pt[i].x==pt[st].x && pt[i].y<pt[st].y)) st = i;
  }
  ans[anses++] = st;
  for (i=st+1; i<=st+n; i++) {
    for (j=anses-1; j; j--) {
      if (orient(pt+ans[j-1], pt+ans[j], pt+i)>=0) break;
      // Should change the above to strictly greater,
      // if you don't want points that lie on the side (not on a vertex) of the hull
      // If you really want them, you might also put an epsilon in orient
    }
    ans[j+1] = i;
```

```
      anses = j+2;
    }
    for (i=0; i<anses; i++) ans[i] = pt[ans[i]].id;
    return anses;
}

int main(void) {
    int i, j, jj;
    double tmp;

    scanf("%d", &n);
    for (i=0; i<n; i++) scanf("%lf %lf", &x[i], &y[i]);
    for (i=0; i<n; i++) {
      x[n] = 2*(-INF)-x[i]; y[n] = y[i];
      x[n+1] = x[i]; y[n+1] = 2*INF-y[i];
      x[n+2] = 2*INF-x[i]; y[n+2] = y[i];
      x[n+3] = x[i]; y[n+3] = 2*(-INF)-y[i];
      for (j=0; j<n+4; j++) if (j!=i) {
        jj = j - (j>i);
        inv[jj].id = j;
        tmp = (x[j]-x[i])*(x[j]-x[i]) + (y[j]-y[i])*(y[j]-y[i]);
        inv[jj].x = (x[j]-x[i])/tmp;
        inv[jj].y = (y[j]-y[i])/tmp;
        inv[jj].ang = atan2(inv[jj].y, inv[jj].x);
      }
      vors = convexHull(n+3, inv, vor);
      // Build bisectors
      for (j=0; j<vors; j++) {
        ans[j][0].x = (x[i]+x[vor[j]])/2;
        ans[j][0].y = (y[i]+y[vor[j]])/2;
        ans[j][1].x = ans[j][0].x - (y[vor[j]]-y[i]);
        ans[j][1].y = ans[j][0].y + (x[vor[j]]-x[i]);
      }
      printf("Around (%lf, %lf)\n", x[i], y[i]);
      // List all intersections of the bisectors
      for (j=1; j<vors; j++) {
        PT vv;
        vv = ComputeLineIntersection(ans[j-1][0], ans[j-1][1],
                                     ans[j][0], ans[j][1]);
        printf("%lf, %lf\n", vv.x, vv.y);
      }
      printf("\n");
    }
    return 0;
}
```

3.8 Usage-1

```
#include "../template.h"
#include "basic.h"
#include "circle.h"
#include "smallestEnclosingCircle.h"

int main()
{
    int test, n;
    double x, y;

    cin >> test;
    while (test--) {
        cin >> n;
        vector<Point> points;
        while (n--) {
            scanf("%lf%lf", &x, &y);
            points.push_back(Point(x, y));
        }

        SmallestEnclosingCircle scc;
        Circle c = scc.getCircle(points);
        printf("%.2lf\n%.2lf %.2lf\n", c.r, c.x, c.y);
    }
}
```

4 Graph - DfsTree

4.1 Biconnected Component

```
// Input graph: vector< vector<int> > a, int n
// Note: 0-indexed
// Usage: BiconnectedComponent bc; (bc.components is the list of components)

struct BiconnectedComponent {
    vector<int> num, low, s;
    vector< vector<int> > components;
```

```
    int counter;

    BiconnectedComponent() : num(n, -1), low(n, -1), counter(0) {
        for (int i = 0; i < n; i++)
            if (num[i] < 0)
                dfs(i, 1);
    }

    void dfs(int x, int isRoot) {
        low[x] = num[x] = ++counter;
        if (a[x].empty()) {
            components.push_back(vector<int>(1, x));
            return;
        }
        s.push_back(x);

        for (int i = 0; i < (int) a[x].size(); i++) {
            int y = a[x][i];
            if (num[y] > -1) low[x] = min(low[x], num[y]);
            else {
                dfs(y, 0);
                low[x] = min(low[x], low[y]);

                if (isRoot || low[y] >= num[x]) {
                    components.push_back(vector<int>(1, x));
                    while (1) {
                        int u = s.back();
                        s.pop_back();
                        components.back().push_back(u);
                        if (u == y) break;
                    }
                }
            }
        }
    }
};
```

4.2 Biconnected Component Check

```
// http://vn.spoj.com/problems/SAFENET2/
// Problem: Output the maximum size of a biconnected component

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

int n;
vector <int> a[30000];

struct BiconnectedComponent {
  vector<int> low, num, s;
  vector< vector<int> > components;
  int counter;

  BiconnectedComponent() : num(n, -1), low(n, -1), counter(0) {
    for (int i = 0; i < n; i++)
      if (num[i] < 0)
        dfs(i, 1);
  }

  void dfs(int x, int isRoot) {
    low[x] = num[x] = ++counter;
    if (a[x].empty()) {
      components.push_back(vector<int>(1, x));
      return;
    }
    s.push_back(x);

    for (int i = 0; i < a[x].size(); i++) {
      int y = a[x][i];
      if (num[y] > -1) low[x] = min(low[x], num[y]);
      else {
        dfs(y, 0);
        low[x] = min(low[x], low[y]);

        if (isRoot || low[y] >= num[x]) {
          components.push_back(vector<int>(1, x));
          while (1) {
            int u = s.back();
            s.pop_back();
            components.back().push_back(u);
            if (u == y) break;
          }
        }
      }
    }
  }
};

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

```
  while (m--)
  {
    scanf("%d%d", &x, &y);
    a[--x].push_back(--y);
    a[y].push_back(x);
  }

  BiconnectedComponent bc;
  int ans = 0;
  for (int i = 0; i < bc.components.size(); i++)
    ans = max(ans, int(bc.components[i].size()));
  printf("%d\n", ans);
}
```

4.3 Articulation Bridge/Points

```
// NOTE: DOES NOT WORK WHEN THERE ARE MULTIPLE EDGES! To fix, use map to count # occurrences of edges
// Assume already have undirected graph vector< vector<int> > G with V vertices
// Vertex index from 0
// Usage:
// UndirectedDfs tree;
// Then you can use tree.bridges and tree.cuts
struct UndirectedDfs {
    vector<int> low, num, parent;
    vector<bool> articulation;
    int counter, root, children;

    vector< pair<int,int> > bridges;
    vector<int> cuts;

    UndirectedDfs() : low(V, 0), num(V, -1), parent(V, 0), articulation(V, false),
            counter(0), children(0) {
        for(int i = 0; i < V; ++i) if (num[i] == -1) {
            root = i; children = 0;
            dfs(i);
            articulation[root] = (children > 1);
        }
        for(int i = 0; i < V; ++i)
            if (articulation[i]) cuts.push_back(i);
    }
private:
    void dfs(int u) {
        low[u] = num[u] = counter++;
        for(int j = 0; j < (int) G[u].size(); ++j) {
            int v = G[u][j];
            if (num[v] == -1) {
                parent[v] = u;
                if (u == root) children++;
                dfs(v);
                if (low[v] >= num[u])
                    articulation[u] = true;
                if (low[v] > num[u]) bridges.push_back(make_pair(u, v));
                low[u] = min(low[u], low[v]);
            } else if (v != parent[u])
                low[u] = min(low[u], num[v]);
        }
    }
};
```

4.4 Articulation Bridge/Points Check

```
#include "../../template.h"

int V;
vector< vector <int> > G;

#include "BridgeArticulation.h"

int main() {
    freopen("input.txt", "r", stdin);
    int m;
    while (cin >> V >> m) {
        G.resize(V);
        REP(i,V) G[i].clear();

        while (m--) {
            int u, v; cin >> u >> v;
            --u; --v;
            G[u].push_back(v);
            G[v].push_back(u);
        }
        UndirectedDfs tree;
        tree.solve();

        cout << "Bridges: ";
        REP(i,tree.bridges.size()) cout << 1+tree.bridges[i].first << ' ' << 1+tree.bridges[i].second << "    ";
```

```
            cout << endl;
            cout << "Cut points: ";
            REP(i,tree.cuts.size()) cout << 1+tree.cuts[i] << ' ';
            cout << endl;
        }
    }
```

4.5 SCC

```
// Assume that already have directed graph vector< vector<int> > G with V vertices
// Index from 0
// Usage:
// DirectedDfs tree;
// Now you can use tree.scc
struct DirectedDfs {
    vector<int> num, low, current, S;
    int counter;
    vector< vector<int> > scc;

    DirectedDfs() : num(V, -1), low(V, 0), current(V, 0), counter(0) {
        REP(i,V) if (num[i] == -1) dfs(i);
    }

    void dfs(int u) {
        low[u] = num[u] = counter++;
        S.push_back(u);
        current[u] = 1;
        REP(j, G[u].size()) {
            int v = G[u][j];
            if (num[v] == -1) dfs(v);
            if (current[v]) low[u] = min(low[u], low[v]);
        }
        if (low[u] == num[u]) {
            scc.push_back(vector<int>());
            while (1) {
                int v = S.back(); S.pop_back(); current[v] = 0;
                scc.back().push_back(v);
                if (u == v) break;
            }
        }
    }
};
```

4.6 SCC Check

```
#include "../../template.h"

int V;
vector< vector<int> > G;

#include "StronglyConnected.h"

int main() {
    freopen("input.txt", "r", stdin);
    int m;
    while (cin >> V >> m) {
        G.resize(V);
        REP(i,V) G[i].clear();

        while (m--) {
            int u, v; cin >> u >> v;
            --u; --v;
            G[u].push_back(v);
        }

        DirectedDfs tree;
        tree.solve();

        DEBUG(V);
        REP(i,tree.scc.size()) {
            REP(j,tree.scc[i].size())
                cout << 1+tree.scc[i][j] << ' ';
            cout << endl;
        }
    }
}
```

5 Graph - Matching

5.1 FastMatching

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

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

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

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

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

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

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

5.2 General Matching

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

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

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

```
void init(int _n) {
    n = _n; E = 0;
    for(int x=1; x<=n; ++x) { last[x] = -1; mat[x] = 0; }
}
void add(int u, int v) {
    adj[E] = v; next[E] = last[u]; last[u] = E++;
}
int lca(int u, int v) {
    for(int x=1; x<=n; ++x) inpath[x] = false;
    while (true) {
        u = root[u];
        inpath[u] = true;
        if (u == start) break;
        u = dad[mat[u]];
    }
    while (true) {
        v = root[v];
        if (inpath[v]) break;
        v = dad[mat[v]];
    }
    return v;
}
void trace(int u) {
    while (root[u] != newRoot) {
        int v = mat[u];

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

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

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

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

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

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

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

5.3 Min cost matching

```
// Index from 1
// Min cost matching
// Usage: init(); for[i,j,cost] addEdge(i, j, cost)
//
// Tested:
// - SGU 210
// - SGU 206

#define arg __arg
long long c[MN][MN];
long long fx[MN], fy[MN];
int mx[MN], my[MN];
int trace[MN], qu[MN], arg[MN];
long long d[MN];
int front, rear, start, finish;

void init() {
    FOR(i,1,N) {
        fy[i] = mx[i] = my[i] = 0;
        FOR(j,1,N) c[i][j] = inf;
    }
}

void addEdge(int i, int j, long long cost) {
    c[i][j] = min(c[i][j], cost);
}

inline long long getC(int i, int j) {
    return c[i][j] - fx[i] - fy[j];
}

void initBFS() {
    front = rear = 1;
    qu[1] = start;
    memset(trace, 0, sizeof trace);
    FOR(j,1,N) {
        d[j] = getC(start, j);
        arg[j] = start;
    }
    finish = 0;
}

void findAugPath() {
    while (front <= rear) {
        int i = qu[front++];
        FOR(j,1,N) if (!trace[j]) {
            long long w = getC(i, j);
            if (!w) {
                trace[j] = i;
                if (!my[j]) {
                    finish = j;
                    return ;
                }
                qu[++rear] = my[j];
            }
            if (d[j] > w) {
                d[j] = w;
                arg[j] = i;
            }
        }
    }
}

void subx_addy() {
    long long delta = inf;
    FOR(j,1,N)
        if (trace[j] == 0 && d[j] < delta) delta = d[j];

    // xoay
    fx[start] += delta;
    FOR(j,1,N)
        if (trace[j]) {
            int i = my[j];
            fy[j] -= delta;
            fx[i] += delta;
        }
        else d[j] -= delta;

    FOR(j,1,N)
        if (!trace[j] && !d[j]) {
            trace[j] = arg[j];
            if (!my[j]) { finish = j; return ; }
            qu[++rear] = my[j];
```

```
        }
    }

    void enlarge() {
        do {
            int i = trace[finish];
            int next = mx[i];
            mx[i] = finish;
            my[finish] = i;
            finish = next;
        } while (finish);
    }

    int mincost() {
        FOR(i,1,N) fx[i] = *min_element(c[i]+1, c[i]+N+1);
        FOR(j,1,N) {
            fy[j] = c[1][j] - fx[1];
            FOR(i,1,N) {
                fy[j] = min(fy[j], c[i][j] - fx[i]);
            }
        }
        FOR(i,1,N) {
            start = i;
            initBFS();
            while (finish == 0) {
                findAugPath();
                if (!finish) subx_addy();
            }
            enlarge();
        }
        int res = 0;
        FOR(i,1,N) res += c[i][mx[i]];
        return res;
    }
```

5.4 Mtaching

```
    // Index from 0
    // Assume 2 sides have same number of vertices
    struct Matching {
        int n;
        vector< vector<int> > ke;
        vector< bool > seen;
        vector< int > matchL, matchR;

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

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

        bool bpm(int u) {
            for(__typeof(ke[u].begin()) v = ke[u].begin(); v != ke[u].end(); ++v) {
                if (seen[*v]) continue;
                seen[*v] = true;

                if (matchR[*v] < 0 || bpm(matchR[*v])) {
                    matchL[u] = *v;
                    matchR[*v] = u;
                    return true;
                }
            }
            return false;
        }

        int match() {
            int res = 0;
            for(int i = 0; i < n; ++i) {
                for(int i = 0; i < n; ++i) seen[i] = false;
                if (bpm(i)) ++res;
            }
            return res;
        }
    };
```

5.5 Perfect Matching Min Cost Hungarian Assignment

```
    // Hungarian Assignment
    // Note:
    // - Vertex indexed from #0
    // - Change INF and int to long long (if needed)
    //
    // Tested:
    // - SGU 210
    // - SGU 206
```

```
class PerfectMatchingMinCost {
private:
    typedef vector<int> VI;
    typedef vector<VI> VII;
    const int INF = 1e9;

    int N;
    VII cost, adj;
    VI d, fx, fy, mx, my, arg, trace;

    queue<int> q;

    int getCost(int x, int y) {
        return cost[x][y] - fx[x] - fy[y];
    }

    void initBFS(int start) {
        for (int i = 0; i < N; i++)
            trace[i] = -1;
        while (!q.empty()) q.pop();
        q.push(start);
        for (int i = 0; i < N; i++)
            d[i] = getCost(start, i), arg[i] = start;
    }

    int findPath() {
        while (!q.empty()) {
            int x = q.front(); q.pop();
            for (int i = 0; i < (int) adj[x].size(); i++) {
                int y = adj[x][i];
                if (trace[y] == -1) {
                    int w = getCost(x, y);
                    if (w == 0) {
                        trace[y] = x;
                        if (my[y] == -1)
                            return y;
                        q.push(my[y]);
                    }
                    if (d[y] > w) {
                        d[y] = w;
                        arg[y] = x;
                    }
                }
            }
        }
        return -1;
    }

    int update(int start) {
        int delta = INF;
        for (int y = 0; y < N; y++)
            if (trace[y] == -1)
                delta = min(delta, d[y]);
        fx[start] += delta;
        for (int y = 0; y < N; y++)
            if (trace[y] != -1) {
                int x = my[y];
                fx[x] += delta;
                fy[y] -= delta;
            }
            else d[y] -= delta;
        for (int y = 0; y < N; y++)
            if (trace[y] == -1 && d[y] == 0) {
                trace[y] = arg[y];
                if (my[y] == -1)
                    return y;
                q.push(my[y]);
            }
        return -1;
    }

    void enlarge(int finish) {
        for (int y = finish; y != -1; ) {
            int x = trace[y];
            int yy = mx[x];
            mx[x] = y;
            my[y] = x;
            y = yy;
        }
    }

public:
    PerfectMatchingMinCost(int n = 0) {
        N = n;
        cost = VII(n, VI(n, INF));
        adj = VII(n);

        trace = VI(n);
        arg = VI(n);
        fx = VI(n, -INF);
        fy = VI(n);
        d = VI(n);
        mx = VI(n, -1);
        my = VI(n, -1);
    }
```

```
    void AddEdge(int x, int y, int c) {
        if (cost[x][y] == INF) adj[x].push_back(y);
        if (cost[x][y] > c) cost[x][y] = c;
    }

    int GetMinCost() {
        for (int x = 0; x < N; x++) {
            initBFS(x);
            int finish = -1;
            do {
                finish = findPath();
                if (finish != -1) break;
                finish = update(x);
            } while (finish == -1);
            enlarge(finish);
        }
        int ret = 0;
        for (int x = 0; x < N; x++)
            ret += cost[x][mx[x]];
        return ret;
    }
};
```

5.6 Stable Marriage

```
/* Numbered from 0
 * For man i, L[i] = list of women in order of decreasing preference
 * For women j, R[j][i] = index of man i in j-th women's list of preference
 * OUTPUTS:
 *      - L2R[]:    the mate of man i (always between 0 and n-1)
 *      - R2L[]:    the mate of woman j (or -1 if single)
 * COMPLEXITY: M^2
 */

#define MAXM 1024
#define MAXW 1024
int m;
int L[MAXM][MAXW], R[MAXW][MAXM];
int L2R[MAXM], R2L[MAXW];
int p[MAXM];
void stableMarriage(){
    static int p[128];
    memset(R2L, -1, sizeof R2L);
    memset(p, 0, sizeof p);
    // Each man proposes...
    for(int i = 0; i < m; i++) {
        int man = i;
        while (man >= 0) { // propose until success
            int wom;
            while (1) {
                wom = L[man][p[man]++];
                if (R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]]) break;
            }
            int hubby = R2L[wom];
            R2L[L2R[man] = wom] = man;
            man = hubby; // remarry the dumped guy
        }
    }
}
```

6 Graph - Max Flow

6.1 Min Cut Between Every Pair Of Vertices

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

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

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

```
    }

    void addEdge(int u, int v, int c) {
        cap[u][v] += c;
    }

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

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

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

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

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

6.2 Max Closure

```
Problem: Given directed G=(V,E) and cost function f: V --> R. Find max (or min) weight closure. (Closure: set of
    vertices with no outgoing edges)
Max closure <--> Min closure: compliment
Max closure <--> Min cut in H, where H is constructed as follow:
- Add source s, sink t
- if f(v) > 0 --> add edge (s, v) with capacity = f(v)
- if f(v) < 0 --> add edge (v, t) with capacity = -f(v)
- All edges in G have infinite capacity in H

Vertices in same side as s forms a closure. Weight(cut) = sum(f(v) where f(v) > 0) - weight(closure) --> cut is minimum
     when closure is maximum

Wiki: https://en.wikipedia.org/wiki/Closure_problem
```

6.3 Dinic

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

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

// Tested:
// - https://open.kattis.com/problems/maxflow
// - https://www.lydsy.com/JudgeOnline/problem.php?id=1001
const int INF = 1000000000;

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

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

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

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

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

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

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

6.4 MaxFlow Trace

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

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

// Tested:
// - https://open.kattis.com/problems/maxflow
const int INF = 1000111000111000111LL;

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

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

```
    vector< vector<int> > g;

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

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

    vector<int> trace() {
        bfs();
        vector<int> res;
        for(auto edge : e) {
            if (d[edge.a] >= 0 && d[edge.b] < 0 && edge.cap > 0) {
                res.push_back(edge.id);
            }
        }
        return res;
    }

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

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

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

6.5 MaxFlow PushRelabel

```
// Push relabel in O(V^2 E^0.5) with gap heuristic
// Source: https://github.com/dacin21/dacin21_codebook/blob/master/flow/maxflow_short.cpp
//
// Notes:
// - Index from 0
// - Does not work with unsigned types.
//
// Usage:
//
// PushRelabel<int> flow(n);
```

```
// flow.addEdge(u, v, f);  // directed
// flow.addEdge(u, v, f, f);  // undirected
// int maxFlow = flow.maxFlow(s, t);
//
// Tested:
// - http://vn.spoj.com/problems/NKFLOW/ (directed)
// - http://vn.spoj.com/problems/FFLOW/  (undirected)
// - http://www.spoj.com/problems/FASTFLOW/  (undirected)
// - https://codeforces.com/problemset/problem/269/C  (with trace).
//
// TLE on https://www.lydsy.com/JudgeOnline/problem.php?id=1001. Why? (ACed with Dinic flow).

template<typename flow_t = long long>
struct PushRelabel {
    struct Edge {
        int to, rev;
        flow_t f, c;
    };
    vector<vector<Edge> > g;
    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

    int addEdge(int s, int t, flow_t cap, flow_t rcap=0) {
        if (s == t) return -1;
        Edge a = {t, (int)g[t].size(), 0, cap};
        Edge b = {s, (int)g[s].size(), 0, rcap};
        g[s].push_back(a);
        g[t].push_back(b);

        // Return ID of forward edge.
        return b.rev;
    }

    flow_t maxFlow(int s, int t) {
        int v = g.size();
        H[s] = v;
        ec[t] = 1;
        vector<int> co(2*v);
        co[0] = v-1;
        for (int i = 0; i < v; ++i) {
            cur[i] = g[i].data();
        }
        for (auto &e : g[s]) {
            add_flow(e, e.c);
        }
        if (hs[0].size()) {
            for (int hi = 0; hi >= 0;) {
                int u = hs[hi].back();
                hs[hi].pop_back();
                while (ec[u] > 0) { // discharge u
                    if (cur[u] == g[u].data() + g[u].size()) {
                        H[u] = 1e9;
                        for (auto &e : g[u]) {
                            if (e.c && H[u] > H[e.to]+1) {
                                H[u] = H[e.to]+1;
                                cur[u] = &e;
                            }
                        }
                        if (++co[H[u]], !--co[hi] && hi < v) {
                            for (int i = 0; i < v; ++i) {
                                if (hi < H[i] && H[i] < v){
                                    --co[H[i]];
                                    H[i] = v + 1;
                                }
                            }
                        }
                        hi = H[u];
                    } else if (cur[u]->c && H[u] == H[cur[u]->to] + 1) {
                        add_flow(*cur[u], min(ec[u], cur[u]->c));
                    }
                    else {
                        ++cur[u];
                    }
                }
                while (hi>=0 && hs[hi].empty()) --hi;
            }
        }
        return -ec[s];
    }

private:
    vector<flow_t> ec;
    vector<Edge*> cur;
    vector<vector<int> > hs;
    vector<int> H;

    void add_flow(Edge& e, flow_t f) {
        Edge &back = g[e.to][e.rev];
        if (!ec[e.to] && f) {
            hs[H[e.to]].push_back(e.to);
        }
        e.f += f; e.c -= f;
        ec[e.to] += f;
        back.f -= f; back.c += f;
        ec[back.to] -= f;
```

```
    }
};
```

6.6 Min Cost Max Flow PushRelabel

```
// Source: https://github.com/dacin21/dacin21_codebook/blob/master/flow/mincost_PRonly.cpp
//
// Notes:
// - Index from 0
// - Costs multiplied by N --> overflow when big cost?
// - Does not work with floating point..
//
// Tested:
// - https://www.infoarena.ro/problema/fmcm
// - https://open.kattis.com/problems/mincostmaxflow
// - http://vn.spoj.com/problems/MINCOST  (trace)

template<typename flow_t = int, typename cost_t = int>
struct MinCostFlow {
    struct Edge {
        cost_t c;
        flow_t f;
        int to, rev;
        Edge(int _to, cost_t _c, flow_t _f, int _rev) : c(_c), f(_f), to(_to), rev(_rev) {}
    };

    int N, S, T;
    vector<vector<Edge> > G;
    MinCostFlow(int _N, int _S, int _T) : N(_N), S(_S), T(_T), G(_N), eps(0) {}

    void addEdge(int a, int b, flow_t cap, cost_t cost) {
                assert(cap >= 0);
        assert(a >= 0 && a < N && b >= 0 && b < N);
        if (a == b) { assert(cost >= 0); return; }
        cost *= N;
        eps = max(eps, abs(cost));
        G[a].emplace_back(b, cost, cap, G[b].size());
        G[b].emplace_back(a, -cost, 0, G[a].size() - 1);
    }

    flow_t getFlow(Edge const &e) {
        return G[e.to][e.rev].f;
    }

    pair<flow_t, cost_t> minCostMaxFlow() {
        cost_t retCost = 0;
        for (int i = 0; i<N; ++i) {
            for (Edge &e : G[i]) {
                retCost += e.c*(e.f);
            }
        }
        //find max-flow
        flow_t retFlow = max_flow();
        h.assign(N, 0); ex.assign(N, 0);
        isq.assign(N, 0); cur.assign(N, 0);
        queue<int> q;
        for (; eps; eps >>= scale) {
            //refine
            fill(cur.begin(), cur.end(), 0);
            for (int i = 0; i < N; ++i) {
                for (auto &e : G[i]) {
                    if (h[i] + e.c - h[e.to] < 0 && e.f) push(e, e.f);
                }
            }
            for (int i = 0; i < N; ++i) {
                if (ex[i] > 0){
                    q.push(i);
                    isq[i] = 1;
                }
            }
            // make flow feasible
            while (!q.empty()) {
                int u = q.front(); q.pop();
                isq[u]=0;
                while (ex[u] > 0) {
                    if (cur[u] == G[u].size()) {
                        relabel(u);
                    }
                    for (unsigned int &i=cur[u], max_i = G[u].size(); i < max_i; ++i) {
                        Edge &e = G[u][i];
                        if (h[u] + e.c - h[e.to] < 0) {
                            push(e, ex[u]);
                            if (ex[e.to] > 0 && isq[e.to] == 0) {
                                q.push(e.to);
                                isq[e.to] = 1;
                            }
                            if (ex[u] == 0) break;
                        }
                    }
                }
            }
```

```
            if (eps > 1 && eps>>scale == 0) {
                eps = 1<<scale;
            }
        }
        for (int i = 0; i < N; ++i) {
            for (Edge &e : G[i]) {
                retCost -= e.c*(e.f);
            }
        }
        return make_pair(retFlow, retCost / 2 / N);
    }

private:
    static constexpr cost_t INFCOST = numeric_limits<cost_t>::max()/2;
    static constexpr int scale = 2;

    cost_t eps;
    vector<unsigned int> isq, cur;
    vector<flow_t> ex;
    vector<cost_t> h;
    vector<vector<int> > hs;
    vector<int> co;

    void add_flow(Edge& e, flow_t f) {
        Edge &back = G[e.to][e.rev];
        if (!ex[e.to] && f) {
            hs[h[e.to]].push_back(e.to);
        }
        e.f -= f; ex[e.to] += f;
        back.f += f; ex[back.to] -= f;
    }

    void push(Edge &e, flow_t amt) {
        if (e.f < amt) amt = e.f;
        e.f -= amt; ex[e.to] += amt;
        G[e.to][e.rev].f += amt; ex[G[e.to][e.rev].to] -= amt;
    }

    void relabel(int vertex){
        cost_t newHeight = -INFCOST;
        for (unsigned int i = 0; i < G[vertex].size(); ++i){
            Edge const&e = G[vertex][i];
            if(e.f && newHeight < h[e.to] - e.c){
                newHeight = h[e.to] - e.c;
                cur[vertex] = i;
            }
        }
        h[vertex] = newHeight - eps;
    }

    flow_t max_flow() {
        ex.assign(N, 0);
        h.assign(N, 0); hs.resize(2*N);
        co.assign(2*N, 0); cur.assign(N, 0);
        h[S] = N;
        ex[T] = 1;
        co[0] = N-1;
        for (auto &e : G[S]) {
            add_flow(e, e.f);
        }
        if (hs[0].size()) {
            for (int hi = 0; hi>=0;) {
                int u = hs[hi].back();
                hs[hi].pop_back();
                while (ex[u] > 0) { // discharge u
                    if (cur[u] == G[u].size()) {
                        h[u] = 1e9;
                        for(unsigned int i = 0; i < G[u].size(); ++i) {
                            auto &e = G[u][i];
                            if (e.f && h[u] > h[e.to]+1) {
                                h[u] = h[e.to]+1, cur[u] = i;
                            }
                        }
                        if (++co[h[u]], !--co[hi] && hi < N) {
                            for (int i = 0; i < N; ++i) {
                                if (hi < h[i] && h[i] < N) {
                                    --co[h[i]];
                                    h[i] = N + 1;
                                }
                            }
                        }
                        hi = h[u];
                    } else if (G[u][cur[u]].f && h[u] == h[G[u][cur[u]].to]+1) {
                        add_flow(G[u][cur[u]], min(ex[u], G[u][cur[u]].f));
                    } else {
                        ++cur[u];
                    }
                }
                while (hi>=0 && hs[hi].empty()) {
                    --hi;
                }
            }
        }
        return -ex[S];
```

```
    }
};
```

6.7 Min Cost Max Flow Shortest Path Faster Algorithm

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

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

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

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

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

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

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

    Flow findFlow(int x, Flow flow) {
        if (x == T) {
            totalCost += dis[S] * flow;
            totalFlow += flow;
            return flow;
        }
```

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

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

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

6.8 Minimum Cut

```
// Minimum cut between every pair of vertices (Stoer Wagner)
pair<int, VI> GetMinCut(VVI &weights) {
    int N = weights.size();
    VI used(N), cut, best_cut;
    int best_weight = -1;

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

6.9 Misc?

```
// Tested:
// - http://codeforces.com/gym/100199/submission/12608232
// - SGU 176

Feasible flow in network with upper + lower constraint, no source, no sink:
- For each edge in original flow:
  - Add edge with cap' = upper bound - lower bound.
- Add source s, sink t.
- Let M[v] = (sum of lowerbounds of ingoing edges to v) - (sum of lower bounds of outgoing edges from v).
- For all v, if M[v] > 0, add (s, v, M), else add (v, t, -M).
- If all outgoing edges from S are full --> feasible flow exists, it is flow + lower bounds.

Feasible flow in network with upper + lower constraint, with source & sink:
- Add edge (t, s) with capacity [0, INF].
- Check feasible in network without source & sink.

Max flow with both upper + lower constraints, source s, sink t: add edge (t, s, +INF).
- Binary search lower bound, check whether feasible flow exists WITHOUT source / sink
```

7 Graph- MST

7.1 Directed MST

```
const int maxe = 100111;
const int maxv = 100;

// Index from 0
// Running time O(E*V)
namespace chuliu {
    struct Cost;
    vector<Cost> costlist;

    struct Cost {
        int id, val, used, a, b, pos;
        Cost() { val = -1; used = 0; }
        Cost(int _id, int _val, bool temp) {
            a = b = -1; id = _id; val = _val; used = 0;
            pos = costlist.size(); costlist.push_back(*this);
        }
        Cost(int _a, int _b) {
            a = _a; b = _b; id = -1; val = costlist[a].val - costlist[b].val;
            used = 0; pos = costlist.size(); costlist.push_back(*this);
        }
        void push() {
            if (id == -1) {
                costlist[a].used += used;
                costlist[b].used -= used;
            }
        }
    };

    struct Edge {
        int u, v;
        Cost cost;
        Edge() {}
        Edge(int id, int _u, int _v, int c) {
            u = _u; v = _v; cost = Cost(id, c, 0);
        }
    } edge[maxe];

    int n, m, root, pre[maxv], node[maxv], vis[maxv], best[maxv];

    void init(int _n) {
        n = _n; m = 0;
        costlist.clear();
    }

    void add(int id, int u, int v, int c) {
        edge[m++] = Edge(id, u, v, c);
    }

    int mst(int root) {
        int ret = 0;

        while (true) {
            REP(i, n) best[i] = -1;

            REP(e, m) {
                int u = edge[e].u, v = edge[e].v;
                if ((best[v] == -1 || edge[e].cost.val < costlist[best[v]].val) && u != v) {
                    pre[v] = u;
                    best[v] = edge[e].cost.pos;
                }
            }

            REP(i, n) if (i != root && best[i] == -1) return -1;

            int cntnode = 0;
            memset(node, -1, sizeof node); memset(vis, -1, sizeof vis);

            REP(i, n) if (i != root) {
                ret += costlist[best[i]].val;
                costlist[best[i]].used++;

                int v = i;
                while (vis[v] != i && node[v] == -1 && v != root) {
                    vis[v] = i;
                    v = pre[v];
                }

                if (v != root && node[v] == -1) {
                    for (int u = pre[v]; u != v; u = pre[u]) node[u] = cntnode;
                    node[v] = cntnode++;
                }
            }

            if (cntnode == 0) break;

            REP(i, n) if (node[i] == -1) node[i] = cntnode++;
```

```
            REP(e, m) {
                int v = edge[e].v;
                edge[e].u = node[edge[e].u];
                edge[e].v = node[edge[e].v];
                if (edge[e].u != edge[e].v) edge[e].cost = Cost(edge[e].cost.pos, best[v]);
            }

            n = cntnode;
            root = node[root];
        }

        return ret;
    }

    vector<int> trace() {
        vector<int> ret;
        FORD(i, costlist.size()-1,0) costlist[i].push();
        REP(i, costlist.size()) {
            Cost cost = costlist[i];
            if (cost.id != -1 && cost.used > 0) ret.push_back(cost.id);
        }
        return ret;
    }
}
```

8 Graph- Misc

8.1 2SAT

```
// Tested:
// - http://codeforces.com/contest/568/problem/C
// - https://open.kattis.com/contests/nwerc15open/problems/cleaningpipes
#include <bits/stdc++.h>
#define REP(i,a) for(int i=0,_a=(a); i<_a; i++)
using namespace std;

const int MN = 200111;   // 2 * no variables.
int n;
vector<int> g[MN], gt[MN];
vector<bool> used;
vector<int> order, comp;

void dfs1 (int v) {
    used[v] = true;
    for (size_t i=0; i<g[v].size(); ++i) {
        int to = g[v][i];
        if (!used[to])
            dfs1 (to);
        }
        order.push_back (v);
}
void dfs2 (int v, int cl) {
    comp[v] = cl;
    for (size_t i=0; i<gt[v].size(); ++i) {
        int to = gt[v][i];
        if (comp[to] == -1)
            dfs2 (to, cl);
        }
}
int main() {
    // n = 2 * (number of boolean variables)
    // NOTE: if we need to fix some variable, e.g. set i = 0 --> addEdge(2*i+1, 2*i)
    // var i --> 2 nodes: 2*i, 2*i+1.

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

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

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

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

```
        }
    for (int i=0; i<n; i += 2) {
        int ans = comp[i] > comp[i^1] ? i : i^1;
        printf ("%d ", ans);
    }
}
```

8.2 Euler Path

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

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

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

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

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

8.3 Euler Path Check

```
#include "template.h"
#include "Graph/Misc/EulerPath.h"

int main() {
    add_edge(1, 2);
    add_edge(2, 3);
    add_edge(3, 4);
    add_edge(4, 1);
    add_edge(1, 3);

    find_path(1);
    PR0(path, path.size());
}
```

8.4 Max Clique

```
// Source: https://github.com/bobogei81123/bcw_codebook/blob/master/codes/Graph/Maximum_Clique/Maximum_Clique.cpp
class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
```

```
    int s[MV][MV/30+1];
    vector<int> sol;

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

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

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

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

9 JAVA

9.1 BigMath

```
import java.math.*;

class BMath {
    static int cnt1, cnt2;
    public static MathContext mc = null;
    public static BigDecimal eps = null;
    public static BigDecimal two = null;
    public static BigDecimal sqrt3 = null;
    public static BigDecimal pi = null;
    public static final int PRECISION = 128;

    static {
        mc = new MathContext(PRECISION);
        eps = BigDecimal.ONE.scaleByPowerOfTen(-PRECISION);
        two = BigDecimal.valueOf(2);
        sqrt3 = sqrt(BigDecimal.valueOf(3));
        pi = asin(BigDecimal.valueOf(0.5)).multiply(BigDecimal.valueOf(6));
    }

    // val > 0
    public static BigInteger sqrt(BigInteger val) {
        int len = val.bitLength();
        BigInteger left = BigInteger.ONE.shiftLeft((len - 1) / 2);
        BigInteger right = BigInteger.ONE.shiftLeft(len / 2 + 1);
```

```
        while (left.compareTo(right) < 0) {
            BigInteger mid = left.add(right).shiftRight(1);
            if (mid.multiply(mid).compareTo(val) <= 0) {
                left = mid.add(BigInteger.ONE);
            } else {
                right = mid;
            }
        }
        return right.subtract(BigInteger.ONE);
    }

    public static BigDecimal sqrt(BigDecimal val) {
        BigInteger unscaledVal = val.scaleByPowerOfTen(2 * mc.getPrecision()).toBigInteger();
        return new BigDecimal(sqrt(unscaledVal)).scaleByPowerOfTen(-mc.getPrecision());
    }

    // arcsin x =    (n=1       ) [(2n)!]x^(2n+1)/[4^n*(n!)^2*(2n+1)]
    // arctan x =    (n=1       ) [(-1)^n]x^(2n+1)/(2n+1)
    public static BigDecimal asin(BigDecimal val) {
        BigDecimal tmp = val;
        BigDecimal ret = tmp;
        val = val.multiply(val, mc);
        for (int n = 1; tmp.compareTo(eps) > 0; ++n) {
            tmp = tmp.multiply(val, mc).multiply(
                    BigDecimal.valueOf(2 * n - 1).divide(BigDecimal.valueOf(2 * n), mc), mc);
            ret = ret.add(tmp.divide(BigDecimal.valueOf(2 * n + 1), mc), mc);
        }
        return ret;
    }
}
```

9.2 IO

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

class InputReader {
    private final BufferedReader reader;
    private StringTokenizer tokenizer;

    public InputReader(InputStream stream) {
        reader = new BufferedReader(new InputStreamReader(stream));
        tokenizer = null;
    }

    public String nextLine() {
        try {
            return reader.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public String next() {
        while (tokenizer == null || !tokenizer.hasMoreTokens()) {
            tokenizer = new StringTokenizer(nextLine());
        }
        return tokenizer.nextToken();
    }

    public int nextInt() {
        return Integer.parseInt(next());
    }
}
```

9.3 Regex Test

```
package Java;
import java.util.regex.*;
public class RegexTest {
    public static void main(String[] args) {
        Pattern p = Pattern.compile("a+");
        String s = "aabbbaaaaabbbabb";
        System.out.println(s.matches("a+b+a+b+a+b+"));
        Matcher m = p.matcher(s);
        while (m.find()) {
            System.out.println(m.start() + "-->" + m.end() + ": " + m.group());
        }
    }
}
```

10 Math - Linear

10.1 Gaussian Jordan Elimination

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

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

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

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

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

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

10.2 Gaussian Binary

```
// Tested: http://codeforces.com/gym/100211 - E
// n : number of rows
// m : number of columns
int gauss (vector < bitset<N> > a, int n, int m, bitset<N> & ans) {
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        for (int i=row; i<n; ++i)
            if (a[i][col]) {
                swap (a[i], a[row]);
                break;
            }
        if (! a[row][col])
            continue;
```

```
            where[col] = row;

            for (int i=0; i<n; ++i)
                if (i != row && a[i][col])
                    a[i] ^= a[row];
            ++row;
        }
        // The rest of implementation is the same as above
    }
```

10.3 Matrix Class Java

```
import java.math.*;
import java.util.*;
import java.io.*;

class Matrix
{
    public int m, n;
    public BigInteger[][] a;

    Matrix(int m, int n) {
        this.m = m;
        this.n = n;
        a = new BigInteger[m][n];
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                a[i][j] = BigInteger.ZERO;
    }

    public void print() {
        System.out.println("size = " + m + " " + n);
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++)
                System.out.print(a[i][j] + " ");
            System.out.println();
        }
    }

    public BigInteger det() {
        boolean rev = false;
        BigInteger mult = BigInteger.ONE;
        for(int j = 0; j < n; j++) {
            int save = -1;
            for(int i = j; i < n; i++)
                if (a[i][j].compareTo(BigInteger.ZERO) != 0)
                if (save < 0 || a[save][j].abs().compareTo(a[i][j].abs()) > 0)
                    save = i;

            if (save < 0 || a[save][j].compareTo(BigInteger.ZERO) == 0) return BigInteger.ZERO;

            if (save != j) {
                rev = !rev;
                for(int k = 0; k < n; k++) {
                    BigInteger tmp = a[j][k];
                    a[j][k] = a[save][k];
                    a[save][k] = tmp;
                }
            }

            BigInteger m1, m2;
            for(int i = j+1; i < n; i++) if (a[i][j].compareTo(BigInteger.ZERO) != 0) {
                m1 = a[j][j];
                m2 = a[i][j];
                mult = mult.multiply(m1);
                for(int k = 0; k < n; k++) {
                    a[i][k] = a[i][k].multiply(m1).subtract(a[j][k].multiply(m2));
                }
            }
        }
        BigInteger res = BigInteger.ONE;
        for(int i = 0; i < n; i++)
            res = res.multiply(a[i][i]);
        if (rev) res = res.negate();
        return res.divide(mult);
    }
}
```

11 Math - NumberThoery

11.1 Find any Prime Factor of n

```
// Find any prime factor of n.
```

```cpp
#include "template.h"
#include "Math/Prime/RabinMiller.h"

long long mul(long long a, long long b, long long mod) {
    if (b == 0) return 0;
    if (b == 1) return a % mod;
    long long mid = mul(a, b >> 1, mod);
    mid = (mid + mid) % mod;
    if (b & 1) return (mid + a) % mod;
    else return mid;
}

long long brent(long long n) {
    if (n == 1) return 1;
    if (!(n & 1)) return 2;
    if (!(n % 3)) return 3;

    const int p[3] = {1, 3, 5};
    long long y, q, x, ys, g, my = 3;
    int i, j, k, m, r, c;

    for (i = 0; i < my; ++i) {
        y = 1; r = 1; q = 1; m = 111; c = p[i];

        do {
            x = y; k = 0;
            for (j = 1; j <= r; ++j) y = (mul(y, y, n) + c) % n;
            do {
                ys = y;
                for (j = 1; j <= min(m, r-k); ++j) {
                    y = (mul(y, y, n) + c) % n;
                    q = mul(q, abs(x - y), n);
                }
                g = __gcd(q, n); k += m;
            } while (k < r && g < 2);
            r <<= 1;
        } while (g < 2);

        if (g == n)
            do {
                ys = (mul(ys, ys, n) + c) % n;
                g = __gcd(abs(x - ys), n);
            } while (g < 2);

        if (g != n) return g;
    }
    return n;
}
```

11.2 Find any Prime Factor of n check

```cpp
#include "../../template.h"
#include "brent.h"

int main() {
    // print 1 2 3 2 5 2 7 2 3 2
    for(int i = 1; i <= 10; ++i)
        cout << brent(i) << ' ';
    cout << endl;

    for(int test = 0; test < 100000; ++test) {
        long long n = rand();
        if (n < 0) n = -n;
        long long x = brent(n);

        assert(n % x == 0);
        if (test % 100 == 0) {
            DEBUG(test);
        }
    }
}
```

11.3 sum(aixi)=b(mod m)

```cpp
// Solve the equation: a1x1 + a2x2 +     + anxn     b (modul m)
// Where a1, a2, ..., an, b, m are positive integers.

int g[MAXN], x[MAXN];

bool congruenceEquation(vector<int> a, int b, int m, vector<int> &ret) {
    int n = sz(a);
    a.pb(m);
    g[0] = a[0];
    For(i, 1, n) g[i] = gcd(g[i - 1], a[i]);
    ret.clear();
    if (b % g[n]) return false;
    int val = b / g[n];
    Ford(i, n, 1) {
```

```
        pair<ll, ll> p = extgcd(g[i - 1], a[i]);
        x[i] = p.se * val % m;
        val = p.fi * val % m;
    }
    x[0] = val;
    For(i, 0, n) x[i] = (x[i] + m) % m;
    Rep(i, n) ret.pb(x[i]);
    return true;
}
```

11.4 Factorial Mod

```
int factmod (int n, int p) { // n!, excluding p^k of course
    int res = 1;
    while (n > 1) {
        res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
        for (int i=2; i<=n%p; ++i)
            res = (res * i) % p;
        n /= p;
    }
    return res % p;
}
```

11.5 Primitive Root

```
// Primitive root of modulo n is integer g iff for all a < n & gcd(a, n) == 1, there exist k: g^k = a mod n
// k is called discrete log of a (in case P is prime, can find in O(sqrt(P)) by noting that (P-1) is divisible by k)
//
// Exist if:
// - n is 1, 2, 4
// - n = p^k for odd prime p
// - n = 2*p^k for odd prime p
int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --b;
        else
            a = int (a * 1ll * a % p),  b >>= 1;
    return res;
}

int generator (int p) {
    vector<int> fact;
    int phi = p-1,  n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

11.6 Sqrt Mod

```
// Jacobi Symbol (m/n), m, n  0  and n is odd
// (m/n)==1    x^2 == m (mod n) solvable, -1 unsolvable
#define NEGPOW(e) ((e) % 2 ? -1 : 1)
int jacobi(int a, int m) {
    if (a == 0) return m == 1 ? 1 : 0;
    if (a % 2)  return NEGPOW((a-1)*(m-1)/4)*jacobi(m%a, a);
    else return NEGPOW((m*m-1)/8)*jacobi(a/2, m);
}
int invMod(int a, int m) {
    int x, y;
    if (extgcd(a, m, x, y) == 1) return (x + m) % m;
    else                                 return 0; // unsolvable
}
// No solution when: n(p-1)/2 = -1 mod p
int sqrtMod(int n, int p) { //find x: x2 = n (mod p) p is prime
    int S, Q, W, i, m = invMod(n, p);
    for (Q = p - 1, S = 0; Q % 2 == 0; Q /= 2, ++S);
```

```
    do { W = rand() % p; } while (W == 0 || jacobi(W, p) != -1);
    for (int R = powMod(n, (Q+1)/2, p), V = powMod(W, Q, p); ;) {
        int z = R * R * m % p;
        for (i = 0; i < S && z % p != 1; z *= z, ++i);
        if (i == 0) return R;
        R = (R * powMod(V, 1 << (S-i-1), p)) % p;
    }
}
int powMod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --b;
        else
            a = int (a * 1ll * a % p),  b >>= 1;
    return res;
}
```

11.7 Number theory (modular, Chinese remainder, linear Diophantine)

```
// 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.

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

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

// 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;
        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.
PII 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.
PII chinese_remainder_theorem(const VI &m, const VI &r) {
        PII 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
        PII 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;
}
```

11.8 Details in Code

```
//Find all x such that x^k=a(mod n) n is pime
int generator (int p) { //Return primirity root of prime p
    vector<int> fact; int phi = p-1,  n = phi;
    for (int i=2; i*i<=n; ++i)
```

```
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)    n /= i;
        }
    if (n > 1)
        fact.push_back (n);
    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powMod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
int main() {
    int n, k, a; cin >> n >> k >> a;
    if (a == 0) { puts ("1\n0");  return 0; }
    int g = generator (n);
    int sq = (int) sqrt (n + .0) + 1;
    vector < pair<int,int> > dec (sq);
    for (int i=1; i<=sq; ++i)
    dec[i-1] = make_pair (powMod (g, int (i * sq * 1ll * k % (n - 1)), n), i);
    sort (dec.begin(), dec.end());
    int any_ans = -1;
    for (int i=0; i<sq; ++i) {
        int my = int (powMod (g, int (i * 1ll * k % (n - 1)), n) * 1ll * a % n);
        vector < pair<int,int> >::iterator it =
                lower_bound (dec.begin(), dec.end(), make_pair (my, 0));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq - i;
            break;
        }
    }
    if (any_ans == -1) { puts ("0");  return 0; }
    int delta = (n-1) / gcd (k, n-1);  vector<int> ans;
    for (int cur=any_ans%delta; cur<n-1; cur+=delta)
        ans.push_back (powMod (g, cur, n));
    sort (ans.begin(), ans.end());
    for (size_t i=0; i<ans.size(); ++i) printf ("%d ", ans[i]);
}
```

12 Math - Prime

12.1 EulerPhi

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

12.2 Sieve

```
typedef unsigned int uint;

// NOTE: gP(n) is incorrect for even values of n
#define N 90000000
uint mark[N / 64 + 1];
#define gP(n) (mark[(n)>>6]&(1<<(((n)>>1)&31)))
```

```
#define rP(n) (mark[(n)>>6]&=~(1<<(((n)>>1)&31)))

// prime indexed from 0
uint prime[5222222], nprime;

void sieve() {
    memset( mark, -1, sizeof( mark ) );
    uint i;
    uint sqrtN = ( uint )sqrt( ( double )N ) + 1;

    for( i = 3; i < sqrtN; i += 2 ) if( gP( i ) ) {
        uint i2 = i + i;
        for( uint j = i * i; j < N; j += i2 ) rP( j );
    }
    nprime = 0;
    prime[nprime++] = 2;
    for( i = 3; i < N; i += 2)
        if (gP(i)) prime[nprime++] = i;
}
```

12.3 PollardRho

```
import java.math.BigInteger;
import java.security.SecureRandom;

class PollardRho {
    private final static BigInteger ZERO = new BigInteger("0");
    private final static BigInteger ONE  = new BigInteger("1");
    private final static BigInteger TWO  = new BigInteger("2");
    private final static SecureRandom random = new SecureRandom();

    public static BigInteger rho(BigInteger N) {
        BigInteger divisor;
        BigInteger c  = new BigInteger(N.bitLength(), random);
        BigInteger x  = new BigInteger(N.bitLength(), random);
        BigInteger xx = x;

        // check divisibility by 2
        if (N.mod(TWO).compareTo(ZERO) == 0) return TWO;

        do {
            x  =  x.multiply(x).mod(N).add(c).mod(N);
            xx = xx.multiply(xx).mod(N).add(c).mod(N);
            xx = xx.multiply(xx).mod(N).add(c).mod(N);
            divisor = x.subtract(xx).gcd(N);
        } while((divisor.compareTo(ONE)) == 0);

        return divisor;
    }

    public static void factor(BigInteger N) {
        if (N.compareTo(ONE) == 0) return;
        if (N.isProbablePrime(20)) { System.out.println(N); return; }
        BigInteger divisor = rho(N);
        factor(divisor);
        factor(N.divide(divisor));
    }

    public static void main(String[] args) {
        BigInteger N = new BigInteger(args[0]);
        factor(N);
    }
}
```

12.4 RabinMiller Prime Test

```
typedef long long ll;

// mulMod and powMod is same as Math/modulo.h.
// These 2 functions are duplicated here for easier copy-paste.

/**
 * When MOD < 2^63, use following mulMod:
 * Source: https://en.wikipedia.org/wiki/Modular_arithmetic#Example_implementations
 * On computer architectures where an extended precision format with at least 64 bits
 * of mantissa is available (such as the long double type of most x86 C compilers),
 * the following routine is faster than any algorithmic solution, by employing the
 * trick that, by hardware, floating-point multiplication results in the most
 * significant bits of the product kept, while integer multiplication results in the
 * least significant bits kept
 */
uint64_t mulMod(uint64_t a, uint64_t b, uint64_t m) {
    long double x;
    uint64_t c;
    int64_t r;
```

```
    if (a >= m) a %= m;
    if (b >= m) b %= m;

    x = a;
    c = x * b / m;
    r = (int64_t)(a * b - c * m) % (int64_t)m;
    return r < 0 ? r + m : r;
}

/** Calculates a^b % m */
uint64_t powMod(uint64_t a, uint64_t b, uint64_t m) {
    uint64_t r = m==1?0:1; // make it works when m == 1.
    while (b > 0) {
        if (b & 1) r = mulMod(r, a, m);
        b = b >> 1;
        a = mulMod(a, a, m);
    }
    return r;
}

bool suspect(ll a, ll s, ll d, ll n) {
    ll x = powMod(a, d, n);
    if (x == 1) return true;
    for (int r = 0; r < s; ++r) {
        if (x == n - 1) return true;
        x = mulMod(x, x, n);
    }
    return false;
}
// {2,7,61,-1}                          is for n < 4759123141 (= 2^32)
// {2,3,5,7,11,13,17,19,23,-1} is for n < 10^15 (at least)
bool isPrime(int64_t n) {
    if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
    ll test[] = {2,3,5,7,11,13,17,19,23,-1};
    ll d = n - 1, s = 0;
    while (d % 2 == 0) ++s, d /= 2;
    for (int i = 0; test[i] < n && test[i] != -1; ++i)
        if (!suspect(test[i], s, d, n)) return false;
    return true;
}
// Killer prime: 5555555557LL (fail when not used mulMod)
```

12.5 Segmented Sieve

```
// table[i-L] == true <=> i == prime
const int SQRTN = 1<<16; // upperbound of sqrt(H) + 10
vector<bool> segmentSieve(ll L, ll H) {
    static ll p[SQRTN];
    static int lookup = 0;
    if (!lookup) {
        for (ll i = 2; i < SQRTN; ++i) p[i] = i;
        for (ll i = 2; i*i < SQRTN; ++i)
            if (p[i])
                for (ll j = i*i; j < SQRTN; j += i)
                    p[j] = 0;
        remove(p, p+SQRTN, 0);
        lookup = 1;
    }
    vector<bool> table(H - L);
    for (ll i = L; i < H; ++i) table[i - L] = 1;
    for (ll i = 0, j; p[i] * p[i] < H; ++i) { // O( \sqrt(H) )
        if (p[i] >= L) j = p[i] * p[i];
        else if (L % p[i] == 0) j = L;
        else j = L - (L % p[i]) + p[i];
        for (; j < H; j += p[i]) table[j-L] = 0;
    }
    return table;
}
```

13 Math - Pure

13.1 Fibonacci Check

```
// Proof : http://www.fq.math.ca/Scanned/10-4/advanced10-4.pdf
bool isSquare(long long n) { /* */}
bool isFibonacci(int n) {
    return n >= 0 && isSquare(5*n*n+4) || isSquare(5*n*n-4);
}
```

13.2 Bernoulli Numbers and Faulhaber's Formula

```
Finding sum(i ^ k)
//Faulhaber's Formula: sum(i ^ k) = 1 / (a + 1) * sum_(j = 0->a) _(-1) ^ j * a + 1_C_j * bernoulli[j] * n ^ (a + 1 - j)
    const int MAX_N = 1e3 + 9;
const int INF = 2000000000;
const ll MOD = 1e9 + 7;
ll combi[MAX_N + 2][MAX_N + 2]; //aCb = combi[a][b]
void makeCombiMod()
{
    combi[0][0] = 1;
    combi[1][0] = 1;
    combi[1][1] = 1;
    for (int i = 2; i <= MAX_N; i++)
    {
        combi[i][0] = 1;
        for (int j = 1; j <= i; j++)
        {
            combi[i][j] = combi[i - 1][j] + combi[i - 1][j - 1];
            combi[i][j] %= MOD;
        }
    }
}

// a x + b y = gcd(a, b)
ll extgcd(ll a, ll b, ll &x, ll &y)
{
    ll g = a;
    x = 1;
    y = 0;
    if (b != 0)
        g = extgcd(b, a % b, y, x), y -= (a / b) * x;
    return g;
}
// 1/a mod m
ll mod_inverse(ll a)
{
    ll x, y;
    extgcd(a, MOD, x, y);
    return (MOD + x % MOD) % MOD;
}

ll B[MAX_N + 2];
void initBernoulliMod()
{
    makeCombiMod();
    for (int i = 0; i < MAX_N + 1; i++)
    {
        if (i % 2)
            B[i] = 0;
        else
            B[i] = -INF;
    }
    B[0] = 1;
    B[1] = -mod_inverse(2) + MOD;
}

ll BernoulliMod(int n)
{
    if (B[n] == -INF)
    {
        ll sum = (1 + combi[n + 1][1] * B[1]) % MOD;
        for (int i = 2; i < n; i += 2)
            sum = (sum + (combi[n + 1][i] * BernoulliMod(i) % MOD)) % MOD;
        B[n] = sum * mod_inverse(n + 1) % MOD;
        B[n] *= -1;
        while (B[n] < 0)
            B[n] += MOD;
    }
    return B[n];
}
ll powMod(ll n, ll p)
{
    ll ans = 1, ln = n % MOD;
    if (p <= 0)
        return 1;
    while (p != 0)
    {
        if ((p & 1) == 1)
            ans = (ans * ln) % MOD;
        ln = (ln * ln) % MOD;
        p = p >> 1;
    }
    return ans;
}
int main()
{
    ll n, a;
    initBernoulliMod();
    QUERY
    {
        scanf("%lld %lld", &n, &a);
        ll ans = 0;
        ll x;
        int i;
        for (int j = 0; j <= a; j++)
```

```
        {
            x = (combi[a + 1][j] * BernoulliMod(j)) % MOD;
            i = a + 1 - j;
            x = (x * powMod(n, i)) % MOD;
            if (j & 1)
                ans -= x;
            else
                ans += x;
            ans %= MOD;
            ans = (ans + MOD) % MOD;
        }
        ll xx = mod_inverse(a + 1);
        ans = (ans * xx) % MOD;
        printf("%lld\n", ans);
    }
}
```

13.3 Moebius

```
If g and f are arithmetic functions satisfying:
    g(n) = sum( f(d) for d|n )
then
    f(n) = sum( mu(d)*g(n/d) for d|n )

where mu(n) is:
    0 if n has squared prime factor
    1 if n is square-free and has even number of prime factor
    -1 if n is square-free and has odd number of prime factor

Generalized version:
    g(n) = sum( f(n div m) for 1 <= m <= n)
then
    f(n) = sum( mu(m) * g(n div m) for 1 <= m <= n)
```

13.4 N Gonal

```
Polygonal numbers: # dots arranged in shape of regular polygon.
3. n*(n+1)/2
4. n*n
5. n*(3*n-1)/2
6. n*(2*n-1)
s. P(s, n) = (n*n*(s-2) - n*(s-4))/2
2*P(s, n) = P(s+k, n) + P(s-k, n)
```

13.5 Pell Equation

```
Consider equation: x*x - n*y*y = 1
Notice that for any solution (x, y), x/y is good approximation of sqrt(n).
In fact, fundamental (which min. x) satisfies:
(x,y) = (hi, ki) where hi/ki is the sequence of convergents to continued fraction for sqrt(n).
From fundamental solution, all solutions can be found using:
x(k+1) = x1*xk + n*y1*yk
y(k+1) = x1*yk + y1*xk
```

13.6 Generate All Primitive Triplets

```
// sinh bo 3 pytago nguyen thuy voi x, y, z <= n
vector< vector<int> > genPrimitivePytTriples(int n) {
    vector< vector<int> > ret;
    for (int r=1; r*r<=n; ++r) for (int s=(r%2==0)?1:2; s<r; s+=2) if (__gcd(r,s)==1) {
        vector<int> t;
        t.push_back(r*r+s*s); //z
        t.push_back(2*r*s); // y
        t.push_back(r*r-s*s); // x
        if (t[0]<=n) ret.push_back(t);
    }
    sort(ret.begin(), ret.end());
    return ret;
}
// a^2 + b^2 == c^2
// To generate all primitive triples:
// a = m^2 - n^2, b = 2mn, c = m^2 + n^2 (m > n)
// Primitive triples iff gcd(m, n) == 1 && (m - n) % 2 == 1
```

13.7 Sum div Sum mod

```
// sumdiv(n) = [n/1] + [n/2] + [n/3] + ... + [n/n]
long long sumdiv(long long n) {
    long long s=0;
    for (long long i=1; i<=n; ) {
        long long q=n/i, r=n/q;
        s+=(long long)(r-i+1)*q;
        i=r+1;
    }
    return s;
}
// summod(n,k) = n%1 + n%2 + ... + n%k    (k<=n)
long long summod(int n, int k) {
    long long s=0;
    for (int i=1; i<=k;) {
        int q=n/i, r=min(k,n/q), t=r-i+1;
        s+=(n%i)*t-(t*(t-1)*q)/2;
        i=r+1;
    }
    return s;
}
```

13.8 Matrix Cpp Class

```
typedef vector<vector<ll>> matrix;
matrix base, I, zero;
int mod = 10;
int K = 2;
matrix mul(matrix A, matrix B)
{
    matrix c(K + 1, vector<ll>(K + 1));
    for (int i = 1; i <= K; i++)
        for (int j = 1; j <= K; j++)
            for (int k = 1; k <= K; k++)
                c[i][j] = (c[i][j] + A[i][k] * B[k][j]) % mod;
    return c;
}
matrix pow(matrix A, int p)
{
    if (p == 0)
        return I;
    if (p == 1)
        return A;
    if (p % 2)
        return mul(A, pow(A, p - 1));
    matrix x = pow(A, p / 2);
    return mul(x, x);
}
matrix sum(matrix A, matrix B)
{
    matrix c(K + 1, vector<ll>(K + 1));
    for (int i = 1; i <= K; i++)
        for (int j = 1; j <= K; j++)
            c[i][j] = (A[i][j] + B[i][j]) % mod;
    return c;
}
matrix bigsum(int p)
{
    if (p == 0)
        return zero;
    if (p % 2 == 1)
        return sum(mul(bigsum(p / 2), sum(I, pow(base, p / 2))), pow(base, p));
    p /= 2;
    return mul(bigsum(p), sum(I, pow(base, p)));
}
int solve()
{
    // create vector F1
    vector<ll> F1(K + 1);
    F1[1] = 1;
    F1[2] = 1;
    // create matrix T
    matrix T(K + 1, vector<ll>(K + 1));
    T[1][1] = 0, T[1][2] = 1;
    T[2][1] = 1, T[2][2] = 1;

    // raise T to the (N-1)th power
    if (N == 1)
        return 1;
    T = pow(T, N - 1);
    // the answer is the first row of T . F1
    ll res = 0;
    REP(i, K)
    res = (res + T[1][i] * F1[i]) % mod;
    return res;
}

Matrix identity(int n) {
    Matrix res(n, n);
    REP(i,n) res.x[i][i] = 1;
    return res;
}
```

13.9 Lehmer Pi

```
// lehmer_pi(n) = number of primes <= n.

const int MAX = 2e6 + 5; //equals 2*sqrt(MAXN) in problem
const int M = 7;         //equals smallest x, st. product of first "x" primes > MAX

vector<int> lp, primes, pi;
int phi[MAX+1][M+1], sz[M+1];

void factor_sieve() {
    lp.resize(MAX);
    pi.resize(MAX);
    lp[1] = 1;
    pi[0] = pi[1] = 0;
    for (int i = 2; i < MAX; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            primes.emplace_back(i);
        }
        for (int j = 0; j < primes.size() && primes[j] <= lp[i]; ++j) {
            int x = i * primes[j];
            if (x >= MAX) break;
            lp[x] = primes[j];
        }
        pi[i] = primes.size();
    }
}

void init() {
    factor_sieve();
    for(int i = 0; i <= MAX; ++i) {
        phi[i][0] = i;
    }
    sz[0] = 1;
    for(int i = 1; i <= M; ++i) {
        sz[i] = primes[i-1]*sz[i-1];
        for(int j = 1; j <= MAX; ++j) {
            phi[j][i] = phi[j][i-1] - phi[j/primes[i-1]][i-1];
        }
    }
}

int sqrt2(long long x) {
    long long r = sqrt(x - 0.1);
    while (r*r <= x) ++r;
    return r - 1;
}

int cbrt3(long long x) {
    long long r = cbrt(x - 0.1);
    while(r*r*r <= x) ++r;
    return r - 1;
}

long long getphi(long long x, int s) {
    if(s == 0)  return x;
    if(s <= M) {
        return phi[x%sz[s]][s] + (x/sz[s])*phi[sz[s]][s];
    }
    if(x <= primes[s-1]*primes[s-1]) {
        return pi[x] - s + 1;
    }
    if(x <= primes[s-1]*primes[s-1]*primes[s-1] && x < MAX) {
        int sx = pi[sqrt2(x)];
        long long ans = pi[x] - (sx+s-2)*(sx-s+1)/2;
        for(int i = s+1; i <= sx; ++i) {
            ans += pi[x/primes[i-1]];
        }
        return ans;
    }
    return getphi(x, s-1) - getphi(x/primes[s-1], s-1);
}

long long getpi(long long x) {
    if(x < MAX)   return pi[x];
    int cx = cbrt3(x), sx = sqrt2(x);
    long long ans = getphi(x, pi[cx]) + pi[cx] - 1;
    for(int i = pi[cx]+1, ed = pi[sx]; i <= ed; ++i) {
        ans -= getpi(x/primes[i-1-1]) - i + 1;
    }
    return ans;
}

long long lehmer_pi(long long x) {
    if(x < MAX)   return pi[x];
    int a = (int)lehmer_pi(sqrt2(sqrt2(x)));
    int b = (int)lehmer_pi(sqrt2(x));
    int c = (int)lehmer_pi(cbrt3(x));
    long long sum = getphi(x, a) + (long long)(b + a - 2) * (b - a + 1) / 2;
    for (int i = a + 1; i <= b; i++) {
        long long w = x / primes[i-1];
```

```
            sum -= lehmer_pi(w);
            if (i > c) continue;
            long long lim = lehmer_pi(sqrt2(w));
            for (int j = i; j <= lim; j++) {
                sum -= lehmer_pi(w / primes[j-1]) - (j - 1);
            }
        }
        return sum;
    }
```

13.10 Knight's Shortest Path

```
    // Knight s shortest path (from (0, 0))
    // Tested:
    // - https://open.kattis.com/problems/knightstrip
    int KSP(int x, int y) {
        if (x < 0) x = -x;
        if (y < 0) y = -y;
        if (x < y) swap(x, y);
        if (x == 1 && y == 0) return 3;
        if (x == 2 && y == 2) return 4;
        int d = x - y;
        if (y > d) return 2 * ((y - d + 2) / 3) + d;
        return d - 2 * ((d - y) / 4);
    }
```

14 String

14.1 Hash

```
    #define MH 600006
    #define Base1 10000019
    #define Base2 10000079
    #define MOD1 1000000007
    #define MOD2 1000000009
    ll B1[MH], B2[MH];
    void init_hash()
    {
        B1[0] = B2[0] = 1;
        for (int i = 1; i < MH; i++)
        {
            B1[i] = (B1[i - 1] * Base1) % MOD1;
            B2[i] = (B2[i - 1] * Base2) % MOD2;
        }
    }
    struct Hash
    {
        pll *H;
        int *digit;
        int len;
        Hash()
        {
            H = new pll[MH];
            digit = new int[MH];
            len = 0;
            H[0] = pll(0, 0);
        }
        void clear()
        {
            len = 0;
            H[0] = pll(0, 0);
        }
        ~Hash()
        {
            delete (H);
            delete (digit);
        }
        void insert(char ch)
        {
            len++;
            digit[len] = ch - 'a' + 1;
            H[len].ff = (((H[len - 1].ff * Base1) % MOD1) + digit[len]) % MOD1;
            H[len].ss = (((H[len - 1].ss * Base2) % MOD2) + digit[len]) % MOD2;
        }
        pll substr(int l, int r)
        {
            if (l > len || r < 1 || r < l)
                return pll(0, 0);
            int sub_len = r - l + 1;
            pll ans;
            ans.ff = (H[r].ff - ((H[l - 1].ff * B1[sub_len]) % MOD1) + MOD1) % MOD1;
            ans.ss = (H[r].ss - ((H[l - 1].ss * B2[sub_len]) % MOD2) + MOD2) % MOD2;
```

```
        return ans;
    }
    pll concate(pll h, int l, int r)
    {
        pll x = substr(l, r);
        int sub_len = r - l + 1;
        h.ff = (((h.ff * B1[sub_len]) % MOD1) + x.ff) % MOD1;
        h.ss = (((h.ss * B2[sub_len]) % MOD2) + x.ss) % MOD2;
        return h;
    }
    bool operator==(const Hash &p) const
    {
        return len == p.len && H[len] == p.H[len];
    }
    pll &operator[](int index)
    {
        return H[index];
    }
};
No array
    string s;
#define Base1 10000019
#define Base2 10000079
#define MOD1 1000000007
#define MOD2 1000000009
map<pll, bool> mp;
pll calc_hash()
{
    int len = s.size();
    ll ans1 = 0;
    ll ans2 = 0;
    for (int i = 0; i < len; i++)
    {
        ll x = s[i] - 'a' + 1;
        ans1 = ((ans1 * Base1) % MOD1 + x) % MOD1;
        ans2 = ((ans2 * Base2) % MOD2 + x) % MOD2;
    }
    return MP(ans1, ans2);
}
const int N = 3 * 1e5 + 9;
pll power[N];
void pre()
{
    power[0].ff = 1;
    power[0].ss = 1;
    for (int i = 1; i < N; i++)
    {
        power[i].ff = (power[i - 1].ff * Base1) % MOD1;
        power[i].ss = (power[i - 1].ss * Base2) % MOD2;
    }
}
pll get_val(int pos, int val)
{
    int px = s.size() - pos - 1;
    px = max(0, px);
    ll div1 = (power[px].ff * (val + 1)) % MOD1;
    ll div2 = (power[px].ss * (val + 1)) % MOD2;
    return MP(div1, div2);
}
```

14.2 SuffixArray

```
// Source: http://codeforces.com/contest/452/submission/7269543
// Efficient Suffix Array O(N*logN)

// String index from 0
// Usage:
// string s;  (s[i] > 0)
// SuffixArray sa(s);
// Now we can use sa.SA and sa.LCP
// sa.LCP[i] = max common prefix suffix of sa.SA[i-1] and sa.SA[i]
struct SuffixArray {
    string a;
    int N, m;
    vector<int> SA, LCP, x, y, w, c;

    SuffixArray(string _a, int m = 256) : a(" " + _a), N(a.length()), m(m),
            SA(N), LCP(N), x(N), y(N), w(max(m, N)), c(N) {
        a[0] = 0;
        DA();
        kasaiLCP();
        #define REF(X) { rotate(X.begin(), X.begin()+1, X.end()); X.pop_back(); }
        REF(SA); REF(LCP);
        a = a.substr(1, a.size());
        for(int i = 0; i < (int) SA.size(); ++i) --SA[i];
        #undef REF
    }

    inline bool cmp (const int a, const int b, const int l) { return (y[a] == y[b] && y[a + l] == y[b + l]); }
```

```
    void Sort() {
        for(int i = 0; i < m; ++i) w[i] = 0;
        for(int i = 0; i < N; ++i) ++w[x[y[i]]];
        for(int i = 0; i < m - 1; ++i) w[i + 1] += w[i];
        for(int i = N - 1; i >= 0; --i) SA[--w[x[y[i]]]] = y[i];
    }

    void DA() {
        for(int i = 0; i < N; ++i) x[i] = a[i], y[i] = i;
        Sort();
        for(int i, j = 1, p = 1; p < N; j <<= 1, m = p) {
            for(p = 0, i = N - j; i < N; i++) y[p++] = i;
            for (int k = 0; k < N; ++k) if (SA[k] >= j) y[p++] = SA[k] - j;
            Sort();
            for(swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i < N; ++i)
                x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1 : p++;
        }
    }

    void kasaiLCP() {
        for (int i = 0; i < N; i++) c[SA[i]] = i;
        for (int i = 0, j, k = 0; i < N; LCP[c[i++]] = k)
            if (c[i] > 0) for (k ? k-- : 0, j = SA[c[i] - 1]; a[i + k] == a[j + k]; k++);
            else k = 0;
    }
};
```

14.3 SuffixArray Check

```
#include "../template.h"
#include "SuffixArray.h"

int main() {
    string a = "abcdabcd";
    SuffixArray sa(a, 256);
    PR0(sa.SA, sa.SA.size()); // 4 0 5 1 6 2 7 3
    PR0(sa.LCP, sa.LCP.size()); // 0 4 0 3 0 2 0 1

    string b = "aaaaaa";
    SuffixArray sb(b, 256);
    PR0(sb.SA, sb.SA.size()); // 5 4 3 2 1 0
    PR0(sb.LCP, sb.LCP.size()); // 0 1 2 3 4 5
}
```

14.4 KMP

```
string t,p;
int b[N];
void kmpPreprocess() {
    int m=p.size();
    int i=0,j=-1;
    b[0]=-1;
    while(i<m) {
        while(j>-1 && p[i]!=p[j])
            j=b[j];
        i++;
        j++;
        b[i]=j;
    }
}
void printFailure(){
    int len=p.size();
    for(int i=0;i<=len;i++){
        printf("%d ",b[i]);
    }
    printf("\n");
}
void kmpSearch() {
    int i=0,j=0;
    int m=p.size();
    int n=t.size();
    while(i<n) {
        while(j>-1 && t[i]!=p[j])
            j=b[j];
        i++;
        j++;
        if(j==m) {
            printf("P is found at index %d in T\n",i-j);
            j=b[j];
        }
    }
}
```

14.5 Palindromic Tree

```
#include "template.h"

const int MAXN = 105000;

struct node {
        int next[26];
        int len;
        int sufflink;
};

int len;
char s[MAXN];
node tree[MAXN];
int num;                        // node 1 - root with len -1, node 2 - root with len 0
int suff;                       // max suffix palindrome

bool addLetter(int pos) {
        int cur = suff, curlen = 0;
        int let = s[pos] - 'a';

        while (true) {
                curlen = tree[cur].len;
                if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
                        break;
                cur = tree[cur].sufflink;
        }
        if (tree[cur].next[let]) {
                suff = tree[cur].next[let];
                return false;
        }

        num++;
        suff = num;
        tree[num].len = tree[cur].len + 2;
        tree[cur].next[let] = num;

        if (tree[num].len == 1) {
                tree[num].sufflink = 2;
                return true;
        }

        while (true) {
                cur = tree[cur].sufflink;
                curlen = tree[cur].len;
                if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
                        tree[num].sufflink = tree[cur].next[let];
                        break;
                }
        }

        return true;
}

void initTree() {
        num = 2; suff = 2;
        tree[1].len = -1; tree[1].sufflink = 1;
        tree[2].len = 0; tree[2].sufflink = 1;
}

int solve() {
        scanf("%s\n", &s[0]);
        len = strlen(s);
        initTree();
        for (int i = 0; i < len; i++) {
                addLetter(i);
        }
        return 0;
}
```

14.6 AhoCorasick

```
// Tested:
// - https://open.kattis.com/problems/stringmultimatching
// Linked list

const int MN = 1000111; // MN > total length of all patterns
struct Node {
    int x; Node *next;
} *nil;
struct List {
    Node *first, *last;
    List() { first = last = nil; }
    void add(int x) {
        Node *p = new Node;
        p->x = x; p->next = nil;
        if (first == nil) last = first = p;
        else last->next = p, last = p;
    }
```

```
};
// End of linked list
struct Aho {
    int qu[MN], suffixLink[MN];
    List leaf[MN];
    int link[MN][MC];
    int sz;

    void init() {
        sz = 0;
        memset(suffixLink, 0, sizeof suffixLink);
        leaf[0] = List();
        memset(link[0], -1, sizeof link[0]);
    }
    int getChild(int type, int v, int c) {
        while (1) {
            if (link[v][c] >= 0) return link[v][c];
            if (type == 1) return 0;
            if (!v) return 0;
            v = suffixLink[v];
        }
    }
    void buildLink() {
        int first, last;
        qu[first = last = 1] = 0;
        while (first <= last) {
            int u = qu[first++];
            REP(c,MC) {
                int v = link[u][c]; if (v < 0) continue;
                qu[++last] = v;
                if (u == 0) suffixLink[v] = 0;
                else suffixLink[v] = getChild(2, suffixLink[u], c);

                if (leaf[suffixLink[v]].first != nil) {
                    if (leaf[v].first == nil) {
                        leaf[v].first = leaf[suffixLink[v]].first;
                        leaf[v].last = leaf[suffixLink[v]].last;
                    }
                    else {
                        leaf[v].last->next = leaf[suffixLink[v]].first;
                        leaf[v].last = leaf[suffixLink[v]].last;
                    }
                }
            }
        }
    }
} aho;
// Usage:
aho.init(); // Initialize
// Foreach query, insert one character at a time:
            int p = 0;
            while (k--) {
                int x; scanf("%d", &x);
                int t = aho.getChild(1, p, x);
                if (t > 0) p = t;
                else {
                    ++aho.sz;
                    aho.leaf[aho.sz] = List();
                    memset(aho.link[aho.sz], -1, sizeof aho.link[aho.sz]);

                    aho.link[p][x] = aho.sz;
                    p = aho.sz;
                }
            }
            aho.leaf[p].add(i);
// Init back link
        aho.buildLink();
// After this stage, we should use aho.getChild(2, node, c) to jump
```

14.7 Find duplicate strings twice a time

```
// Find duplicate strings twice a time
vector<int> z_function(const string & s) {
        int n = (int) s.length();
        vector<int> z(n);
        for (int i = 1, l = 0, r = 0; i < n; ++i) {
                if (i <= r)
                        z[i] = min(r - i + 1, z[i - l]);
                while (i + z[i] < n && s[z[i]] == s[i + z[i]])
                        ++z[i];
                if (i + z[i] - 1 > r)
                        l = i, r = i + z[i] - 1;
        }
        return z;
}

void output_tandem(const string & s, int shift, bool left, int cntr, int l,
                int l1, int l2) {
        int pos;
        if (left) pos = cntr - l1;
```

```
        else pos = cntr - l1 - l2 - l1 + 1;
        cout << "[" << shift + pos << ".." << shift + pos + 2 * l - 1 << "] = "
                        << s.substr(pos, 2 * l) << endl;
}

void output_tandems(const string & s, int shift, bool left, int cntr, int l,
                int k1, int k2) {
        for (int l1 = 1; l1 <= l; ++l1) {
                if (left && l1 == l)
                        break;
                if (l1 <= k1 && l - l1 <= k2)
                        output_tandem(s, shift, left, cntr, l, l1, l - l1);
        }
}

inline int get_z(const vector<int> & z, int i) {
        return 0 <= i && i < (int) z.size() ? z[i] : 0;
}

void find_tandems(string s, int shift = 0) {
        int n = (int) s.length();
        if (n == 1)
                return;

        int nu = n / 2, nv = n - nu;
        string u = s.substr(0, nu), v = s.substr(nu);
        string ru = string(u.rbegin(), u.rend()), rv = string(v.rbegin(), v.rend());

        find_tandems(u, shift);
        find_tandems(v, shift + nu);

        vector<int> z1 = z_function(ru), z2 = z_function(v + '#' + u), z3 =
                        z_function(ru + '#' + rv), z4 = z_function(v);
        for (int cntr = 0; cntr < n; ++cntr) {
                int l, k1, k2;
                if (cntr < nu) {
                        l = nu - cntr;
                        k1 = get_z(z1, nu - cntr);
                        k2 = get_z(z2, nv + 1 + cntr);
                } else {
                        l = cntr - nu + 1;
                        k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
                        k2 = get_z(z4, (cntr - nu) + 1);
                }
                if (k1 + k2 >= l) output_tandems(s, shift, cntr < nu, cntr, l, k1, k2);
        }
}
```

14.8 Suffix Automata

```
struct Node {
    int len, link; // len = max length of suffix in this class
    int next[33];
};
Node s[MN * 2];
set< pair<int,int> > order; // in most application we'll need to sort by len
struct Automaton {
    int sz, last;
    Automaton() {
                order.clear();
        sz = last = 0;
        s[0].len = 0;
        s[0].link = -1;
        ++sz;
        // need to reset next if necessary
    }
    void extend(char c) {
        c = c - 'A';
        int cur = sz++, p;
        s[cur].len = s[last].len + 1;
        order.insert(make_pair(s[cur].len, cur));

        for(p = last; p != -1 && !s[p].next[c]; p = s[p].link)
            s[p].next[c] = cur;
        if (p == -1) s[cur].link = 0;
        else {
            int q = s[p].next[c];
            if (s[p].len + 1 == s[q].len) s[cur].link = q;
            else {
                int clone = sz++;
                s[clone].len = s[p].len + 1;
                memcpy(s[clone].next, s[q].next, sizeof(s[q].next));
                s[clone].link = s[q].link;
                order.insert(make_pair(s[clone].len, clone));

                for(; p != -1 && s[p].next[c] == q; p = s[p].link)
                    s[p].next[c] = clone;
                s[q].link = s[cur].link = clone;
            }
        }
        last = cur;
    }
```

```
};
// Construct:
// Automaton sa; for(char c : s) sa.extend(c);
// 1. Number of distinct substr:
//    - Find number of different paths --> DFS on SA
//    - f[u] = 1 + sum( f[v] for v in s[u].next
// 2. Number of occurrences of a substr:
//    - Initially, in extend: s[cur].cnt = 1; s[clone].cnt = 0;
//    - for(it : reverse order)
//        p = nodes[it->second].link;
//        nodes[p].cnt += nodes[it->second].cnt
// 3. Find total length of different substrings:
//    - We have f[u] = number of strings starting from node u
//    - ans[u] = sum(ans[v] + d[v] for v in next[u])
// 4. Lexicographically k-th substring
//    - Based on number of different substring
// 5. Smallest cyclic shift
//    - Build SA of S+S, then just follow smallest link
// 6. Find first occurrence
//    - firstpos[cur] = len[cur] - 1, firstpos[clone] = firstpos[q]
```

14.9 Manacher

```
// http://vn.spoj.com/problems/PALINY/

#include <bits/stdc++.h>
using namespace std;
const char DUMMY = '.';

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

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

        int len = (f[i] + i % 2) / 2 * 2 + 1 - i % 2;
        if (len > res) {
            res = len;
            center = i;
        }
    }

    return res;
}

int main() {
    int n;
    char s[100100];
    scanf("%d%s", &n, s);
    cout << manacher(string(s, n)) << endl;
}
```

14.10 Z algorithm

```
// z[i] =       d i  x u con d i  nh  t  b  t     u  t    v   tr  i m  tr ng  v  i   o   n     u  c  a  v[]
vector<int> zfunc(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

14.11 Z Algorithm Check

```
#include "../template.h"
#include "zfunc.h"

int main() {
    vector<int> z;
    zfunc("abcabcabc", z); // 9 0 0 6 0 0 3 0 0
    PR0(z, z.size());
    zfunc("aaaaaaaaa", z); // 9 8 7 6 5 4 3 2 1
    PR0(z, z.size());
    return 0;
}
```

15 MISC

15.1 Kth Permutation

```
static public void findPermutation(int n, int k)
{
    int[] numbers = new int[n];
    int[] indices = new int[n];

    // initialise the numbers 1, 2, 3...
    for (int i = 0; i < n; i++)
        numbers[i] = i + 1;

    int divisor = 1;
    for (int place = 1; place <= n; place++)
    {
        if((k / divisor) == 0)
            break;  // all the remaining indices will be zero

        // compute the index at that place:
        indices[n-place] = (k / divisor) % place;
        divisor *= place;
    }

    // print out the indices:
    // System.out.println(Arrays.toString(indices));

    // permute the numbers array according to the indices:
    for (int i = 0; i < n; i++)
    {
        int index = indices[i] + i;

        // take the element at index and place it at i, moving the rest up
        if(index != i)
        {
            int temp = numbers[index];
            for(int j = index; j > i; j--)
               numbers[j] = numbers[j-1];
            numbers[i] = temp;
        }
    }

    // print out the permutation:
    System.out.println(Arrays.toString(numbers));
}
```

15.2 Next Palindrome

```
string ans;
string mid_inc(string t)
{
    int i = (t.size() - 1) / 2;
    int j = i + 1;
    int c = 1;
    if (t[i] == '9')
    {
        c = 1;
        t[i] = '0';
    }
    else
    {
        c = 0;
        t[i] += 1;
    }
    i--;
    if (t.size() % 2 == 0)
    {
        if (t[j] == '9')
        {
            c = 1;
            t[j] = '0';
        }
        else
```

```
            {
                c = 0;
                t[j] = t[j] + 1;
            }
            j++;
        }
        while (i > -1 && j < t.size() && c)
        {
            if (t[i] == '9')
            {
                c = 1;
                t[i] = t[j] = '0';
            }
            else
            {
                c = 0;
                t[i] += 1;
                t[j] = t[i];
            }
            i--;
            j++;
        }
        return t;
    }

    bool check_9(string s)
    {
        for (int i = 0; i < s.size(); i++)
        {
            if (s[i] - '0' != 9)
                return 0;
        }
        ans = "1";
        for (int i = 1; i < s.size(); i++)
            ans += '0';
        ans += '1';
        return 1;
    }
    bool is_palindrome(string t)
    {
        int i = 0, j = t.size() - 1;
        for (; i < j; i++, j--)
        {
            if (t[i] != t[j])
                return 0;
        }
        ans = mid_inc(t);
        return 1;
    }
    void next_palindrome(string t)
    {
        if (check_9(t))
        {
            cout << ans << endl;
            return;
        }
        else if (is_palindrome(t))
        {
            cout << ans << endl;
            return;
        }
        int i = t.size() / 2 - 1;
        int j;
        if (t.size() % 2)
            j = t.size() / 2 + 1;
        else
            j = t.size() / 2;
        while (i > -1 && t[i] == t[j])
            i--, j++;
        bool bigger = 0;
        if (t[i] > t[j])
            bigger = 1;
        while (i > -1)
        {
            t[j] = t[i];
            j++;
            i--;
        }
        if (!bigger)
            ans = mid_inc(t);
        else
            ans = t;
        cout << ans << endl;
    }
    int main()
    {
        string s;
        QUERY
        {
            ans = "";
            cin >> s;
            printf("Case %d: ", _T);
            next_palindrome(s);
        }
```

```
}
```

15.3 Date Cpp

```
// 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;
}
```

15.4 Date Java

```
// Example of using Java's built-in date calculation routines

import java.text.SimpleDateFormat;
import java.util.*;

public class Dates {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        SimpleDateFormat sdf = new SimpleDateFormat("M/d/yyyy");
        while (true) {
            int n = s.nextInt();
            if (n == 0) break;
            GregorianCalendar c = new GregorianCalendar(n, Calendar.JANUARY, 1);
            while (c.get(Calendar.DAY_OF_WEEK) != Calendar.SATURDAY)
                c.add(Calendar.DAY_OF_YEAR, 1);
            for (int i = 0; i < 12; i++) {
                System.out.println(sdf.format(c.getTime()));
                while (c.get(Calendar.MONTH) == i) c.add(Calendar.DAY_OF_YEAR, 7);
            }
        }
    }
}
```

15.5 Template

```
#include <bits/stdc++.h>
using namespace std;
/*--Policy Based Data Structure--*/
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename T>
using ordered_set_rev = tree<T, null_type, greater<T>, rb_tree_tag, tree_order_statistics_node_update>;
#define fbo find_by_order //index wise op
#define ook order_of_key  //number of elements less than key

typedef long long ll;
typedef unsigned long long ull;
#define PI acos(-1.0)
#define eps 1e-9
#define inf 0x3f3f3f3f

#define sf(x) scanf("%I64d", &x)
#define sff(x, y) scanf("%I64d %I64d", &x, &y)
#define sl(x) scanf("%lld", &x)
#define sll(x, y) scanf("%lld %lld", &x, &y)

#define max_ull 18446744073709551615
#define max_ll 9223372036854775807

#define min3(a, b, c) min(a, min(b, c))
#define max3(a, b, c) max(a, max(b, c))
#define min4(a, b, c, d) min(min(a, b), min(c, d))
#define max4(a, b, c, d) max(max(a, b), max(c, d))
#define max5(a, b, c, d, e) max(max3(a, b, c), max(d, e))
#define min5(a, b, c, d, e) min(min3(a, b, c), min(d, e))

#define segtree lt = 2 * par, rg = 2 * par + 1, mid = (st + en) / 2

#define lead_zero(x) __builtin_clzll(x)
#define trail_zero(x) __builtin_ctz(x)
#define total_1s(x) __builtin_popcount(x)
#define first_1(x) __builtin_ffs(x)
#define log2_(x) __builtin_clz(1) - __builtin_clz(x);
#define isPowerOfTwo(x) (x != 0 && (x & (x - 1)) == 0)
#define isLeap(x) ((x % 400 == 0) || (x % 100 ? x % 4 == 0 : false))

#define set(N, cur) N = (N | (1 << cur))
#define reset(N, cur) N = (N & (~(1 << cur)))
#define check(N, cur) ((N & (1 << cur)) == 0)

#define TEST            \
    int test;           \
    scanf("%d", &test); \
    for (int _T = 1; _T <= test; _T++)
#define rep(i, begin, end) for (__typeof(end) i = (begin) - ((begin) > (end)); i != (end) - ((begin) > (end)); i += 1 - 2 * ((begin) > (end)))

/*-----STL------*/
#define fast ios_base::sync_with_stdio(0), cin.tie(0)
#define ii int, int
#define all(v) v.begin(), v.end()
#define reunique(v) v.resize(std::unique(all(v)) - v.begin())
#define pii pair<ii>
#define ff first
#define ss second
#define Iterator(a) __typeof__(a.begin())
#define MERGE(v1, v2, v) merge(all(v1), all(v2), back_inserter(v))
#define MP make_pair
#define PB push_back
#define EB emplace_back

template <typename T>
using dijkstra = priority_queue<T, vector<T>, greater<T>>;

/*____Debug____*/
#define read freopen("input.txt", "r", stdin)
#define write freopen("output.txt", "w", stdout)

#define what_is(x) cerr << #x << " is " << x << endl;

#define error(args...)                              \
    {                                               \
        string _s = #args;                          \
        replace(_s.begin(), _s.end(), ',', ' '); \
        stringstream _ss(_s);                       \
        istream_iterator<string> _it(_ss);          \
        err(_it, args);                             \
    }

void err(istream_iterator<string> it)
{
```

```
}
template <typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args)
{
    cerr << *it << " = " << a << endl;
    err(++it, args...);
}

/* Direction arrays */
/* int dx[] = {1,-1,0,0}, dy[] = {0,0,1,-1};    */                        // 4 Direction
/* int dx[] = {1,-1,0,0,1,1,-1,-1} , dy[] = {0,0,1,-1,1,-1,1,-1}; */    // 8 Direction
/* int dx[] = {1,-1,1,-1,2,2,-2,-2} , dy[] = {2,2,-2,-2,1,-1,1,-1}; */ // Knight Direction
/* int dx[] = {2,-2,1,1,-1,-1} , dy[] = {0,0,1,-1,1,-1}; */             // Hexagonal Direction
/* int dx[] = {2,-2,1,1,-1,-1} , dy[] = {0,0,1,-1,1,-1}; */             // Hexagonal Direction
/* int day[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};*/

int main()
{
}
```