

Mục lục

1	Simple Max Matching
2	Konig
3	Max matching min cost
4	Ganeral Matching
5	Stable Marriage
6	Dinic MaxFlow
7	Mincost MaxFlow SPFA
8	Upper Lower
9	Alternative Tree
10	Max Clique
11	Euler Path
12	Interection of two paths
13	Tree ISO
14	Centroid
15	BiConComps
16	Aho Corasick
17	Suffix Array
18	SuffixAutomata
19	Manacher
20	DP knuth
21	Convex Hull
22	Geometry 2D
23	Geometry 3D
24	C++ tricks

25	FFT	15
26	FFT mod	16
27	NTT	18
28	Gauss	19
29	Simplex	19
30	Chinese Remainder	21
31	Primitive Root	21
32	Range Prime Counting	21
33	Knight's shortest path	22
34	Extended Euclid	22
35	Factorial Mod	22
36	Sqrt Mod	22
37	Interval line	23
38	BIT 2D	23
39	Heavy-Light Decomposition	24
10	1 Simple Max Matching	
11	bool dfs(int u) {	
11	if (mx[u] == T) return false;	
12	mx[u] = T;	
12	for(int v : ke[u]) {	
12	if (!my[v] dfs(my[v])) {	
13	my[v] = u;	
13	return true;	
13	}	
13	}	
13	return false;	
13	}	
14	int main() {	
14	For(i,1,n) {	
15	T++;	
15	res += dfs(i);	

```

    }
    // choose my & i
}

```

2 Konig

```

void konig(){
    queue<int> qu;
    f1(i,m) if (!Assigned[i]) qu.push(i);
    f1(i,n) if (!Assigned[N-i]) qu.push(N-i);
    while (qu.size()){
        int u=qu.front(); qu.pop();
        for (int i=0; int v=a[u][i]; i++)
            if (!(Choosed[v]++)) qu.push(Assigned[v]);
    }
    f1(i,m) if (Assigned[i] && !Choosed[i] && !Choosed[Assigned[i]])
        Choosed[i]=true;
}

```

3 Max matching min cost

```

// numbered from 0. i -> mx[i]
const int V = 1000, INF = 1e9;
int g[V][V], mx[V], my[V], fx[V], fy[V], d[V], ar[V], tr[V], p;
int slack(int u, int v) {
    return g[u][v] - fx[u] - fy[v];
}
int augment(int s) {
    queue<int> q;
    q.push(s);
    fill_n(tr, p, -1);
    for(int i = 0; i < p; ++i) d[i] = slack(s, i), ar[i] = s;
    while(true) {
        while(!q.empty()) {
            int u = q.front();
            q.pop();
            for(int v = 0; v < p; ++v) if(tr[v] == -1) {
                int w = slack(u, v);
                if(w == 0) {
                    tr[v] = u;
                    if(my[v] == -1) return v;
                    q.push(my[v]);
                }
            }
        }
    }
}

```

```

    }
    if(d[v] > w) d[v] = w, ar[v] = u;
}
}
int delta = INF;
for(int v = 0; v < p; ++v) if(tr[v] == -1) delta =
    min(delta, d[v]);
fx[s] += delta;
for(int v = 0; v < p; ++v)
    if(tr[v] == -1) d[v] -= delta;
    else fx[my[v]] += delta, fy[v] -= delta;
for(int v = 0; v < p; ++v) if(tr[v] == -1 && d[v] == 0) {
    tr[v] = ar[v];
    if(my[v] == -1) return v;
    q.push(my[v]);
}
}
}
void maxMatchMinCost() {
    fill_n(mx, p, -1);
    fill_n(my, p, -1);
    for(int i = 0; i < p; ++i) fx[i] = *min_element(g[i], g[i]+p);
    for(int s = 0; s < p; ++s) {
        int f = augment(s);
        while(f != -1) {
            int x = tr[f], nx = mx[x];
            mx[x] = f;
            my[f] = x;
            f = nx;
        }
    }
}
}

```

4 Ganeral Matching

```

class MatchingGraph {
public:
    vector <vector<int> > adj;
    vector <bool> blossom;
    vector <int> parent;
    vector <int> base;
    vector <int> match;
    int n;
}

```

```

MatchingGraph() {
    n = 0;
}

void addEdge(int x, int y) {
    adj[x].push_back(y);
    adj[y].push_back(x);
}

void clearGraph() {
    int i;
    for (i=0; i<SZ(adj); ++i)
        adj[i].clear();
    fill(blossom.begin(), blossom.end(), false);
    fill(parent.begin(), parent.end(), -1);
    for (i=0; i<n; ++i)
        base[i] = i;
    for (i=0; i<n; ++i)
        match[i] = -1;
}

void setN(int newn) {
    n = newn;
    adj.resize(n);
    blossom.resize(n);
    base.resize(n);
    match.resize(n);
    parent.resize(n);
    clearGraph();
}

int lca(int x, int y) {
    vector<bool> fy;
    fy.resize(n);
    fill(fy.begin(), fy.end(), false);
    while (true) {
        x = base[x];
        fy[x] = true;
        if (match[x] == -1)
            break;
        x = parent[match[x]];
    }
    while (true) {
        y = base[y];
        if (fy[y])
            return y;
        y = parent[match[y]];
    }
}

```

```

        return -1;
    }

    void path(int now, int child, int curbase) {
        while (base[now] != curbase) {
            blossom[base[now]] = blossom[base[match[now]]] = true;
            parent[now] = child;
            child = match[now];
            now = parent[match[now]];
        }
    }

    int augmentPath(int x) {
        int i, j;
        for (i=0; i<n; ++i)
            base[i] = i;
        for (i=0; i<n; ++i)
            parent[i] = -1;
        queue<int> bfs;
        vector<bool> sudah;
        sudah.resize(n);
        fill(sudah.begin(), sudah.end(), false);
        sudah[x] = true;
        bfs.push(x);
        while (!bfs.empty()) {
            int now = bfs.front();
            bfs.pop();
            for (i=0; i<SZ(adj[now]); ++i) {
                int next = adj[now][i];
                if (base[next]==base[now] || match[next] == now);
                else if (next == x || (match[next]!=-1 &&
                    parent[match[next]]!=-1)) {
                    int curbase = lca(now, next);
                    fill(blossom.begin(), blossom.end(), false);
                    path(now, next, curbase);
                    path(next, now, curbase);
                    for (j = 0; j < n; ++j)
                        if (blossom[j]) {
                            base[j] = curbase;
                            if (!sudah[j]) {
                                sudah[j] = true;
                                bfs.push(j);
                            }
                        }
                } else if (parent[next]==-1) {
                    parent[next] = now;

```

```

        if (match[next] == -1)
            return next;
        sudah[match[next]] = true;
        bfs.push(match[next]);
    }
}
}
return -1;
}

int edmondsMatch() {
    int i;
    int res = 0;
    for (i=0; i<n; ++i) {
        if (match[i]==-1) {
            int x = augmentPath(i);
            while (x>=0) {
                int p = parent[x];
                int pp = match[p];
                match[x] = p;
                match[p] = x;
                x = pp;
            }
        }
    }
    for (i=0; i<n; ++i)
        if (match[i]!=-1)
            ++res;
    return res >> 1;
}
};

```

5 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 Dinic MaxFlow

```

class DinicFlow {
private:
    vector<int> dist, head, work;
    vector<int> point, flow, capa, next;
    int n, m;

    bool bfs(int s, int t) {
        For(i, 1, n) dist[i] = -1;
        queue<int> q;
        dist[s] = 0;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();

```

```

        for (int i = head[u]; i >= 0; i = next[i])
            if (flow[i] < capa[i] && dist[point[i]] < 0) {
                dist[point[i]] = dist[u] + 1;
                q.push(point[i]);
            }
    }
    return dist[t] >= 0;
}

int dfs(int s, int t, int f) {
    if (s == t) return f;
    for (int &i = work[s]; i >= 0; i = next[i])
        if (flow[i] < capa[i] && dist[point[i]] == dist[s] + 1) {
            int d = dfs(point[i], t, min(f, capa[i] - flow[i]));
            if (d > 0) {
                flow[i] += d;
                flow[i ^ 1] -= d;
                return d;
            }
        }
    return 0;
}

public:
    DinicFlow(int n = 0) {
        this->n = n;
        this->m = 0;
        dist.assign(n + 7, 0);
        head.assign(n + 7, -1);
        work.assign(n + 7, 0);
    }

    void addEdge(int u, int v, int c1, int c2 = 0) {
        point.push_back(v);
        capa.push_back(c1);
        flow.push_back(0);
        next.push_back(head[u]);
        head[u] = m++;
        point.push_back(u);
        capa.push_back(c2);
        flow.push_back(0);
        next.push_back(head[v]);
        head[v] = m++;
    }
}

```

```

int maxFlow(int s, int t) {
    int totFlow = 0;
    while (bfs(s, t)) {
        For(i, 1, n) work[i] = head[i];
        while (true) {
            int d = dfs(s, t, cmax);
            if (d == 0) break;
            totFlow += d;
        }
    }
    return totFlow;
}

```

7 Mincost MaxFlow SPFA

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;

```
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;
}
};

```

8 Upper Lower

- For each edge in original flow:
 - Add edge with cap = upper bound - lower bound.
- Add source s, sink t.
- Let $M[v] = (\text{sum of lowerbounds of ingoing edges to } v) - (\text{sum of lower bounds of outgoing edges from } v)$.
- 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

9 Alternative Tree

```
int n, m, l, q, t, res, test,
    a[maxn], tin[maxn], tout[maxn], mark[maxn], terror[maxn], f[maxn][20];
vector<int> adj[maxn], _adj[maxn];
stack<int> stk;
void visit(const int &u) {
    tin[u] = ++t;
    for(int i = 1; i <= l; ++i) f[u][i] = f[f[u][i-1]][i-1];
    for(auto v : adj[u])
        if (v != f[u][0]) {
            f[v][0] = u;
            visit(v);
        }
    tout[u] = ++t;
}
bool anc(const int &u, const int &v) {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v) {
    if (anc(u,v)) return u;
    if (anc(v,u)) return v;
```

```
    for(int i = 1; i >= 0; --i)
        if (!anc(f[u][i],v)) u = f[u][i];
    return f[u][0];
}
void query() {
    cin >> m;
    for(int i = 1; i <= m; ++i) {
        cin >> a[i];
        _adj[a[i]].clear();
        mark[a[i]] = test;
        terror[a[i]] = test;
    }
    sort(a+1,a+m+1,cmp);
    for(int i = 1; i < m; ++i) {
        int tmp = lca(a[i],a[i+1]);
        if (mark[tmp] < test) {
            mark[tmp] = test;
            a[++m] = tmp;
            _adj[tmp].clear();
        }
    }
    // sort theo tin
    sort(a+1,a+m+1,cmp);
    while (!stk.empty()) stk.pop();
    stk.push(a[1]);
    for(int i = 2; i <= m; ++i) {
        while (tout[stk.top()] < tout[a[i]]) stk.pop();
        _adj[stk.top()].push_back(a[i]);
        stk.push(a[i]);
    }
    res = 0;
    check(a[1]);
    cout << res << "\n";
}
int main() {
    l = log2(n);
    cin >> q;
    f[1][0] = 1;
    visit(1);
    for(test = 1; test <= q; ++test) query();
}
```

10 Max Clique

```

class MaxClique {
public:
    static const int MV = 210;
    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
    int s[MV][MV/30+1];
    vector<int> sol;
    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }
    /* Zero Base */
    void addEdge(int u, int v) {
        if(u > v) swap(u, v);
        if(u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }
    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for(int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if(k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if(c == 0) {
            if(k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for(int i=0; i<(V+31)/32; i++) {
            for(int a = s[k][i]; a ; d++) {
                if(k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while(lb!=1) {
                    lb = (unsigned int)(lb) >> 1;

```

```

                    lg ++;
                }
                int u = i*32 + lg;
                if(k + dp[u] <= ans) return 0;
                if(dfs(u, k+1)) {
                    sol.push_back(v);
                    return 1;
                }
            }
        }
        return 0;
    }
    int solve() {
        for(int i=V-1; i>=0; i--) {
            dfs(i, 1);
            dp[i] = ans;
        }
        return ans;
    }
};

```

11 Euler Path

NOTES:

- When choosing starting vertex (for calling find_path), make sure $\deg[\text{start}] > 0$.
- If find Euler path, starting vertex must have odd degree.
- Check no solution: $SZ(\text{path}) == n\text{Edge} + 1$.
- If directed:
 - Edge $-->$ int
 - add_edge(int a, int b) { adj[a].push_back(b); }
 - Check for no solution:
 - - for all u, $|\text{in_deg}[u] - \text{out_deg}[u]| \leq 1$
 - - At most 1 vertex with $\text{in_deg}[u] - \text{out_deg}[u] = 1$
 - - At most 1 vertex with $\text{out_deg}[u] - \text{in_deg}[u] = 1$ (start vertex)
 - - BFS from start vertex, all vertices u with $\text{out_deg}[u] > 0$ 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;
}

```

12 Interection of two paths

```

int intersect(int a, int b, int c, int d){
    if(lca(b,c)!=c) return 0;
    int z = lca(b,d);
    if(lv[c]<lv[a]){
        if(lca(a,z)==a) return dist(z,a);
    }else{
        if(lca(c,z)==c) return dist(c,z);}
    return 0;
}

```

13 Tree ISO

```

namespace TreeISO {
typedef vector<vector<int>> vvi;
typedef vector<int> vi;
typedef pair<vi, int> pvii;
const int MAXN = 4010;
#define ii pair<int, int>
int N;
vvi edges[2], levels[2];

```

```

int ts[MAXN], label[2][MAXN], parent[2][MAXN];
vi centroid[2];
int findCentroid(const int tID, const int u, const int p) {
    int children = 0, curr;
    for (auto &e : edges[tID][u]) {
        if (e != p) {
            curr = findCentroid(tID, e, u);
            if (curr > (N >> 1))
                break;
            children += curr;
        } //if
    } //for
    if (N - children - 1 <= (N >> 1))
        centroid[tID].push_back(u);
    return ts[u] = children + 1;
} //findCentroid
int setLevels(const int tID, const int u, const int p, const int d) {
    parent[tID][u] = p;
    levels[tID][d].push_back(u);
    int mx = d;
    for (auto &e : edges[tID][u])
        if (e != p)
            mx = max(mx, setLevels(tID, e, u, d + 1));
    return mx;
} //setLevels
bool isoCheck(const int lvl) {
    for (int it = lvl; it >= 0; it--) {
        vector<pvii> order[2];
        for (int i = 0; i < 2; i++) {
            for (auto &u : levels[i][it]) {
                order[i].push_back(pvii(vi(), u));
                for (auto &e : edges[i][u])
                    if (e != parent[i][u])
                        order[i].back().first.push_back(label[i][e]);
            } //for
        } //for
        if ((int) order[0].size() != ((int) order[1].size()))
            return 0;
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < (int) order[0].size(); j++)
                sort(order[i][j].first.begin(), order[i][j].first.end());
            sort(order[i].begin(), order[i].end());
        } //for
        int labelID = 0;

```

```

for (int i = 0; i < (int) order[0].size(); i++) {
    if (order[0][i].first != order[1][i].first)
        return 0;
    if (i && order[0][i].first == order[0][i - 1].first) {
        label[0][order[0][i].second] = label[1][order[1][i].second] =
            labelID;
        continue;
    } //if
    label[0][order[0][i].second] = label[1][order[1][i].second] = ++
        labelID;
    } //for
} //for
return 1;
} //isoCheck
int checkISO(int _N, vector<ii> _edges) {
    N = _N;
    int u, v;
    int T = 1;
    while (T--) {
        int cur = 0;
        memset(ts, 0, sizeof(int) * (N + 2));
        for (int i = 0; i < 2; i++) {
            edges[i].assign(N + 5, vi());
            levels[i].assign(N + 5, vi());
            memset(label[i], 0, sizeof(int) * (N + 2));
            memset(parent[i], 0, sizeof(int) * (N + 2));
            centroid[i].clear();
            for (int j = 0; j < N - 1; j++) {
                int u = _edges[cur].first;
                int v = _edges[cur].second;
                cur++;
                edges[i][u].push_back(v);
                edges[i][v].push_back(u);
            } //for
            findCentroid(i, edges[i][0].empty() ? 1 : 0, -1);
        } //for
        if (edges[0][0].empty())
            N++;
        if ((int) centroid[0].size() != (int) centroid[1].size()) {
            return 0;
        } //if
        if ((int) centroid[0].size() == 2) {
            for (int i = 0; i < 2; i++) {
                for (int j = 0; j < 2; j++) {

```

```

                    edges[i][centroid[i][j]].erase(std::remove(edges[i][centroid[i][
                        j]].begin(),
                            edges[i][centroid[i][
                                j]].end(),
                                    centroid[i][j]),
                                        edges[i][centroid
                                            [i][j]].end());

                    edges[i][centroid[i][j]].push_back(N);
                    edges[i][N].push_back(centroid[i][j]);
                } //for
                centroid[i][0] = N;
            } //for
        } //if
        int d[2];
        for (int i = 0; i < 2; i++)
            d[i] = setLevels(i, centroid[i][0], -1, 0);
        if (d[0] != d[1]) {
            return 0;
        } //if
        if (d[0] >= 0)
            return isoCheck(d[0] - 1) ? 1 : 0;
    } //while
    return 0;
} //main
}

```

14 Centroid

```

void findCentroid(int u, int par, int Size) {
    nChild[u] = 1;
    bool pre = true;
    for (int i = 0; i < a[u].size(); i++) {
        int v = a[u][i];
        if (v != par && ok[v]) {
            findCentroid(v, u, Size);
            if (nChild[v] > Size / 2) pre = false;
            nChild[u] += nChild[v];
        }
    }
    if (pre && nChild[u] >= Size / 2)
        centroid = u;
}

```

15 BiConComps

```

const int N = 1024;

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

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

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

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

```

16 Aho Corasick

```

const int NODE = (int) 1e6 + 1;
const int NC = 26;
int nextNode[NODE][NC];
int chr[NODE];
int parent[NODE];
int prefix[NODE];
int numNodes;
set<int> match[NODE];
int getPrefix(int);
int go(int u, int c) {
    if (nextNode[u][c] != -1) return nextNode[u][c];
    if (u == 0) return 0;
    return nextNode[u][c] = go(getPrefix(u), c);
}
int getPrefix(int u) {
    if (prefix[u] != -1) return prefix[u];
    if (u == 0 || parent[u] == 0) return prefix[u] = 0;
    return prefix[u] = go(getPrefix(parent[u]), chr[u]);
}
void add(const string &s, int id) {
    int u = 0;
    for (int i = 0; i < (int) s.size(); ++i) {
        int c = s[i] - 'A';
        if (nextNode[u][c] == -1) {
            nextNode[u][c] = numNodes;
            fill(nextNode[numNodes], nextNode[numNodes] + NC, -1);
            chr[numNodes] = c;
            parent[numNodes] = u;
            prefix[numNodes] = -1;
            match[numNodes].clear();
            match[numNodes].insert(-1);
            ++numNodes;
        }
        u = nextNode[u][c];
    }
    match[u].insert(id);
}
set<int>& getMatch(int u) {
    if (match[u].count(-1) == 0) return match[u];
    const set<int> &foo = getMatch(getPrefix(u));
    match[u].insert(foo.begin(), foo.end());
    match[u].erase(-1);
}

```

```

    return match[u];
}

void init() {
    fill(nextNode[0], nextNode[0] + NC, -1);
    numNodes = 1;
}

```

17 Suffix Array

```

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
#define REP(i, n) for (int i = 0; i < (int)(n); ++i)
namespace SuffixArray
{
    const int MAXN = 1 << 21;
    char * S;
    int N, gap;
    int sa[MAXN], pos[MAXN], tmp[MAXN], lcp[MAXN];
    bool sufCmp(int i, int j)
    {
        if (pos[i] != pos[j])
            return pos[i] < pos[j];
        i += gap;
        j += gap;
        return (i < N && j < N) ? pos[i] < pos[j] : i > j;
    }
    void buildSA()
    {
        N = strlen(S);
        REP(i, N) sa[i] = i, pos[i] = S[i];
        for (gap = 1;; gap *= 2)
        {
            sort(sa, sa + N, sufCmp);
            REP(i, N - 1) tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i + 1]);
            REP(i, N) pos[sa[i]] = tmp[i];
            if (tmp[N - 1] == N - 1) break;
        }
    }
    void buildLCP()
    {
        for (int i = 0, k = 0; i < N; ++i) if (pos[i] != N - 1)

```

```

        {
            for (int j = sa[pos[i] + 1]; S[i + k] == S[j + k];)
                ++k;
            lcp[pos[i]] = k;
            if (k)--k;
        }
    }
} // end namespace SuffixArray

```

18 SuffixAutomata

```

struct SuffixAutomaton {
    vector<map<char,int>> edges; // edges[i] : the labeled edges from
                                // node i
    vector<int> link;           // link[i] : the parent of i
    vector<int> length;         // length[i] : the length of the longest
                                // string in the ith class
    int last;                   // the index of the equivalence class of
                                // the whole string
    SuffixAutomaton(string s) {
        edges.push_back(map<char,int>());
        link.push_back(-1);
        length.push_back(0);
        last = 0;
        for(int i=0;i<s.size();i++) {
            edges.push_back(map<char,int>());
            length.push_back(i+1);
            link.push_back(0);
            int r = edges.size() - 1;
            int p = last;
            while(p >= 0 && edges[p].find(s[i]) == edges[p].end()) {
                edges[p][s[i]] = r;
                p = link[p];
            }
            if(p != -1) {
                int q = edges[p][s[i]];
                if(length[p] + 1 == length[q]) {
                    link[r] = q;
                } else {
                    edges.push_back(edges[q]);
                    length.push_back(length[p] + 1);
                    link.push_back(link[q]);
                    int qq = edges.size()-1;

```

```

        link[q] = qq;
        link[r] = qq;
        while(p >= 0 && edges[p][s[i]] == q) {
            edges[p][s[i]] = qq;
            p = link[p];
        }
    }
}
last = r;
}
};

```

19 Manacher

```

void manacher() {
    memset(p,0,sizeof p);
    int center = 0, right = 0,mi;
    for (int i = 1; i < n; i++) {
        mi = 2 * center - i;
        if (right > i) p[i] = min(right - i, p[mi]);
        while (a[i+(1+p[i])] == a[i-(1+p[i])]) p[i]++;
        //printf("%d:%d\n",i,p[i]);
        if (i + p[i] > right) {
            right = i+p[i];
            center = i;
        }
    }
}

```

20 DP knuth

<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:

$A[i][j-1] \leq A[i][j] \leq A[i+1][j]$

with $A[i][j]$ = smallest k that gives optimal answer

Also applicable if the following conditions are met:

1. $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$ (quadrangle inequality)
2. $C[b][c] \leq C[a][d]$ (monotonicity)

for all $a \leq b \leq c \leq d$

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;
}

```

21 Convex Hull

struct Point {

long long x, y;

bool operator < (const Point &v) const {

return x == v.x ? y < v.y : x < v.x;

}

long long cross(const Point &p, const Point &q) const {

```

        return (p.x - x) * (q.y - y) - (p.y - y) * (q.x - x);
    }
};

vector<Point> convexHull(vector<Point> p) {
    sort(p.begin(), p.end());
    int k = 0, n = p.size();
    vector<Point> poly (2 * n);
    for(int i = 0; i < n; ++i) {
        while(k >= 2 && poly[k-2].cross(poly[k-1], p[i]) < 0) --k;
        poly[k++] = p[i];
    }
    for(int i = n-2, t = k+1; i >= 0; --i) {
        while(k >= t && poly[k-2].cross(poly[k-1], p[i]) < 0) --k;
        poly[k++] = p[i];
    }
    poly.resize(min(n, max(0, k - 1)));
    return poly;
}

```

22 Geometry 2D

```

// Circle Circle Intersection
// zz: pairs of points
zz circleLine(double r, double a, double b, double c){
    zz res = zz(ii(-1e9 - 1, -1e9 - 1), ii(-1e9 - 1, -1e9 - 1));
    double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b) + eps)
        return res;
    else if (abs (c*c - r*r*(a*a+b*b)) < eps) {
        res.first = ii(x0, y0);
        return res; }
    else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax, ay, bx, by;
        ax = x0 + b * mult; bx = x0 - b * mult;
        ay = y0 - a * mult; by = y0 + a * mult;
        res.first = ii(ax, ay); res.second = ii(bx, by);
        return res; } }

zz circleCircleIntersection(Circle c1, Circle c2) {
    zz res = zz(ii(-1e9 - 1, -1e9 - 1), ii(-1e9 - 1, -1e9 - 1));
    if (dist(ii(c1.x, c1.y), ii(c2.x, c2.y)) < eps) {
        if (abs(c1.r - c2.r) < eps)

```

```

        return res;
    }
    return res; }

double dx = c2.x - c1.x; double dy = c2.y - c1.y;
double A = -2 * dx; double B = -2 * dy;
double C = dx * dx + dy * dy + c1.r * c1.r - c2.r * c2.r;
res = circleLine(c1.r, A, B, C);
res.first = ii(res.first.first + c1.x, res.first.second + c1.y);
res.second = ii(res.second.first + c1.x, res.second.second + c1.y);
return res;
}

///// 2 segments intersection
bool onSegment(Point p, Point q, Point r) // q lies on (p, r)
int orientation(Point p, Point q, Point r){
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2;}

bool doIntersect(Point p1, Point q1, Point p2, Point q2){
    int o1 = orientation(p1, q1, p2); int o2 = orientation(p1, q1, q2);

    ;
    int o3 = orientation(p2, q2, p1); int o4 = orientation(p2, q2, q1);
    if (o1 != o2 && o3 != o4) return true;
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;
    return false;}

////

```

23 Geometry 3D

```

typedef double T;
struct p3 {
    T x,y,z;
    p3 operator+(p3 p) {return {x+p.x, y+p.y, z+p.z};}
    p3 operator-(p3 p) {return {x-p.x, y-p.y, z-p.z};}
    p3 operator*(T d) {return {x*d, y*d, z*d};}
    p3 operator/(T d) {return {x/d, y/d, z/d};} //only for floating-point
    bool operator==(p3 p) {return tie(x,y,z) == tie(p.x,p.y,p.z);}
    bool operator!=(p3 p) {return !operator==(p);}
    T operator|(p3 v, p3 w) {return v.x*w.x + v.y*w.y + v.z*w.z;} //dot
        product
    p3 operator*(p3 v, p3 w) { //cross product
        return {v.y*w.z - v.z*w.y, v.z*w.x - v.x*w.z, v.x*w.y - v.y*w.x};
    }
}

```

```

    }}
T sq(p3 v) {return v|v;}
double abs(p3 v) {return sqrt(sq(v));}
p3 unit(p3 v) {return v/abs(v);}
double angle(p3 v, p3 w) {
    double cosTheta = (v|w) / abs(v) / abs(w);
    return acos(max(-1.0, min(1.0, cosTheta)));}
T orient(p3 p, p3 q, p3 r, p3 s) {return (q-p)*(r-p)|(s-p);} // S vs plane
    PQR
struct plane {
    p3 n; T d; // From normal n and offset d
    plane(p3 n, T d) : n(n), d(d) {} // From normal n and point P
    plane(p3 n, p3 p) : n(n), d(n|p) {} // From three non-collinear points
        P,Q,R
    plane(p3 p, p3 q, p3 r) : plane((q-p)*(r-p), p) {}
    // - these work with T = int
    T side(p3 p) {return (n|p)-d;}
    double dist(p3 p) {return abs(side(p))/abs(n);}
    plane translate(p3 t) {return {n, d+(n|t)};}
    // - these require T = double
    plane shiftUp(double dist) {return {n, d + dist*abs(n)};}
    p3 proj(p3 p) {return p - n*side(p)/sq(n);}
    p3 refl(p3 p) {return p - n*2*side(p)/sq(n);}
}
struct line3d {
    p3 d, o;
    // From two points P, Q
    line3d(p3 p, p3 q) : d(q-p), o(p) {}
    // From two planes p1, p2 (requires T = double)
    line3d(plane p1, plane p2) {
        d = p1.n*p2.n;
        o = (p2.n*p1.d - p1.n*p2.d)*d/sq(d);
    }
    // - these work with T = int
    double sqDist(p3 p) {return sq(d*(p-o))/sq(d);}
    double dist(p3 p) {return sqrt(sqDist(p));}
    bool cmpProj(p3 p, p3 q) {return (d|p) < (d|q);}
    // - these require T = double
    p3 proj(p3 p) {return o + d*(d|(a-o))/sq(d);}
    p3 refl(p3 p) {return proj(p)*2 - p;}
    p3 inter(plane p) {return o - d*p.side(o)/(p.n|d);}
}
double dist(line l1, line l2) {
    p3 n = l1.d*l2.d;
    if (n == zero) return l1.dist(l2.o);
    return abs((l2.o-l1.o)|n)/abs(n);}

```

```

p3 closestOnL1(line l1, line l2) {
    p3 n2 = l2.d*(l1.d*l2.d);
    return l1.o + l1.d*((l2.o-l1.o)|n2)/(l1.d|n2);}
double smallAngle(p3 v, p3 w) {
    return acos(min(abs(v|w)/abs(v)/abs(w), 1.0));}
double angle(plane p1, plane p2) {
    return smallAngle(p1.n, p2.n);}
bool isParallel(plane p1, plane p2) {
    return p1.n*p2.n == zero;}
bool isPerpendicular(plane p1, plane p2) {
    return (p1.n|p2.n) == 0;}
double angle(line3d l1, line3d l2) {
    return smallAngle(l1.p, l2.d);}
bool isParallel(line3d l1, line3d l2) {
    return l1.d*l2.d == zero;}
bool isPerpendicular(line3d l1, line3d l2) {
    return (l1.d|l2.d) == 0;}
double angle(plane p, line3d l) {
    return M_PI/2 - smallAngle(p.n, l.d);}
bool isParallel(plane p, line3d l) {
    return (p.n|l.d) == 0;}
bool isPerpendicular(plane p, line3d l) {
    return p.n*l.d == zero;}
line3d perpThrough(plane p, p3 o) {return line(o, o+p.n);}
plane perpThrough(line3d l, p3 o) {return plane(l.d, o);}

```

24 C++ tricks

```

int __builtin_clz(int x); //number of leading zero
int __builtin_ctz(int x); //number of trailing zero
int __builtin_clzll(long long x); //number of leading zero
int __builtin_ctzll(long long x); //number of trailing zero
int __builtin_popcount(int x); // number of 1-bits in x
int __builtin_popcountll(long long x); //number of 1-bits i

```

25 FFT

```

const double PI = acos(-1.0);
typedef complex<double> Complex;
#define MASK(i) (1LL<<(i))
#define BIT(x,i) (((x) >> (i)) & 1)
#define LOG 17

```

```

Complex fftRoot[MASK(LOG)], invRoot[MASK(LOG)];
#define REP(i, n) for (int i = 0, _n = (n); i < _n; i = i + 1)
void initFFT(void) {
    REP(i, MASK(LOG)) {
        double alpha = 2 * PI / MASK(LOG) * i;
        fftRoot[i] = Complex(cos(alpha), sin(alpha));
        invRoot[i] = Complex(cos(-alpha), sin(-alpha));
    }
}

unsigned roundUp(unsigned v) {
    --v;
    REP(i, 5) v |= v >> MASK(i);
    return v + 1;
}

int reverse(int num, int lg) {
    int res = 0;
    REP(i, lg) if (BIT(num, i)) res |= MASK(lg - i - 1);
    return res;
}

vector<Complex> fft(vector<Complex> a, bool invert) {
    int n = a.size(), lg = 0;
    while (MASK(lg) < n) lg++;
    vector<Complex> roots(n);
    REP(i, n) roots[i] = invert ? invRoot[MASK(LOG) / n * i] :
        fftRoot[MASK(LOG) / n * i];

    REP(i, n) {
        int rev = reverse(i, lg);
        if (i < rev) swap(a[i], a[rev]);
    }

    for (int len = 2; len <= n; len <= 1)
        for (int i = 0; i < n; i += len)
            for (int j = 0; j < (len >> 1); j++) {
                Complex u = a[i + j], v = a[i + j + (len >> 1)] *
                    roots[n / len * j];

                a[i + j] = u + v;
                a[i + j + (len >> 1)] = u - v;
            }

    if (invert) REP(i, n) a[i] /= n;
    return a;
}

vector<long long> multiply(const vector<int> &a, const vector<int>
    &b) {
    int n = roundUp(size(a) + size(b) - 1);
    vector<Complex> pa(n), pb(n);

```

```

    for(int i = 0; i < size(a); ++i) pa[i] = a[i];
    for(int i = 0; i < size(b); ++i) pb[i] = b[i];
    pa = fft(pa, false);
    pb = fft(pb, false);
    for(int i = 0; i < n; ++i) pa[i] *= pb[i];
    pa = fft(pa, true);
    vector<long long> res(n);
    for(int i = 0; i < n; ++i) res[i] = round(real(pa[i]));
    return res;
}

```

26 FFT mod

```

struct cp {
    double x, y;
    cp(double x = 0, double y = 0) : x(x), y(y) {}
    cp operator+(const cp& rhs) const { return cp(x + rhs.x, y + rhs.y); }
    cp operator-(const cp& rhs) const { return cp(x - rhs.x, y - rhs.y); }
    cp operator*(const cp& rhs) const {
        return cp(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x);
    }
    cp operator!() const { return cp(x, -y); }
} rts[maxf + 1];
cp fa[maxf], fb[maxf];
cp fc[maxf], fd[maxf];
int bitrev[maxf];
void fftinit() {
    int k = 0;
    while ((1 << k) < maxf) k++;
    bitrev[0] = 0;
    for (int i = 1; i < maxf; i++) {
        bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << k - 1);
    }
    double PI = acos((double)-1.0);
    rts[0] = rts[maxf] = cp(1, 0);
    for (int i = 1; i + i <= maxf; i++) {
        rts[i] = cp(cos(i * 2 * PI / maxf), sin(i * 2 * PI / maxf));
    }
    for (int i = maxf / 2 + 1; i < maxf; i++) {
        rts[i] = !rts[maxf - i];
    }
}

void dft(cp a[], int n, int sign) {

```



```

static int isinit;
if (!isinit) {
    isinit = 1;
    fftinit();
}
int d = 0;
while ((1 << d) * n != maxf) d++;
for (int i = 0; i < n; i++) {
    if (i < (bitrev[i] >> d)) {
        swap(a[i], a[bitrev[i] >> d]);
    }
}
for (int len = 2; len <= n; len <= 1) {
    int delta = maxf / len * sign;
    for (int i = 0; i < n; i += len) {
        cp *x = a + i, *y = a + i + (len >> 1), *w = sign > 0 ? rts : rts +
            maxf;
        for (int k = 0; k + k < len; k++) {
            cp z = *y * *w;
            *y = *x - z, *x = *x + z;
            x++, y++, w += delta;
        }
    }
}
if (sign < 0) {
    for (int i = 0; i < n; i++) {
        a[i].x /= n;
        a[i].y /= n;
    }
}
void multiply(int a[], int b[], int na, int nb, long long c[], int dup =
    0) {
    int n = na + nb - 1;
    while (n != (n & -n)) n += n & -n;
    for (int i = 0; i < n; i++) fa[i] = fb[i] = cp();
    for (int i = 0; i < na; i++) fa[i] = cp(a[i]);
    for (int i = 0; i < nb; i++) fb[i] = cp(b[i]);
    dft(fa, n, 1);
    if (dup) {
        for (int i = 0; i < n; i++) fb[i] = fa[i];
    } else {
        dft(fb, n, 1);
    }
}

```

```

for (int i = 0; i < n; i++) fa[i] = fa[i] * fb[i];
dft(fa, n, -1);
for (int i = 0; i < n; i++) c[i] = (long long)floor(fa[i].x + 0.5);
}
void multiply(int a[], int b[], int na, int nb, int c[], int mod = (int)1
    e9 + 7,
    int dup = 0) {
    int n = na + nb - 1;
    while (n != (n & -n)) n += n & -n;
    for (int i = 0; i < n; i++) fa[i] = fb[i] = cp();
    static const int magic = 15;
    for (int i = 0; i < na; i++)
        fa[i] = cp(a[i] >> magic, a[i] & (1 << magic) - 1);
    for (int i = 0; i < nb; i++)
        fb[i] = cp(b[i] >> magic, b[i] & (1 << magic) - 1);
    dft(fa, n, 1);
    if (dup) {
        for (int i = 0; i < n; i++) fb[i] = fa[i];
    } else {
        dft(fb, n, 1);
    }
    for (int i = 0; i < n; i++) {
        int j = (n - i) % n;
        cp x = fa[i] + !fa[j];
        cp y = fb[i] + !fb[j];
        cp z = !fa[j] - fa[i];
        cp t = !fb[j] - fb[i];
        fc[i] = (x * t + y * z) * cp(0, 0.25);
        fd[i] = x * y * cp(0, 0.25) + z * t * cp(-0.25, 0);
    }
    dft(fc, n, -1), dft(fd, n, -1);
    for (int i = 0; i < n; i++) {
        long long u = ((long long)floor(fc[i].x + 0.5)) % mod;
        long long v = ((long long)floor(fd[i].x + 0.5)) % mod;
        long long w = ((long long)floor(fd[i].y + 0.5)) % mod;
        c[i] = ((u << magic) + v + (w << magic + magic)) % mod;
    }
}
vector<int> multiply(vector<int> a, vector<int> b, int mod = (int)1e9 + 7)
{
    static int fa[maxf], fb[maxf], fc[maxf];
    int na = a.size(), nb = b.size();
    for (int i = 0; i < na; i++) fa[i] = a[i];
    for (int i = 0; i < nb; i++) fb[i] = b[i];
}

```

```

multiply(fa, fb, na, nb, fc, mod, a == b);
int k = na + nb - 1;
vector<int> res(k);
for (int i = 0; i < k; i++) res[i] = fc[i];
return res;
}

int fpow(int a, int k, int p) {
    if (!k) return 1;
    int res = a, t = a;
    k--;
    while (k) {
        if (k & 1) res = (long long)res * t % p;
        t = (long long)t * t % p;
        k >>= 1;
    }
    return res;
}

vector<int> invert(vector<int> a, int n, int mod) {
    assert(a[0] != 0);
    vector<int> x(1, fpow(a[0], mod - 2, mod));
    while (x.size() < n) {
        vector<int> tmp(a.begin(), a.begin() + min(a.size(), 2 * x.size()));
        vector<int> nx = multiply(multiply(x, x, mod), tmp, mod);
        x.resize(2 * x.size());
        for (int i = 0; i < x.size(); i++) {
            x[i] += x[i];
            x[i] -= nx[i];
            if (x[i] < 0) x[i] += mod;
            if (x[i] >= mod) x[i] -= mod;
        }
    }
    x.resize(n);
    return x;
}

pair<vector<int>, vector<int>> divmod(vector<int> a, vector<int> b, int
mod) {
    int n = a.size(), m = b.size();
    if (n < m) {
        return make_pair(vector<int>(), a);
    }
    reverse(a.begin(), a.end());
    reverse(b.begin(), b.end());
    vector<int> rb = invert(b, n - m + 1, mod);
    vector<int> d = multiply(a, rb, mod);

```

```

reverse(a.begin(), a.end());
reverse(b.begin(), b.end());
while (d.size() > n - m + 1) d.pop_back();
reverse(d.begin(), d.end());
vector<int> r = multiply(d, b, mod);
while (r.size() >= m) r.pop_back();
for (int i = 0; i < m; i++) {
    r[i] = a[i] - r[i];
    if (r[i] < 0) r[i] += mod;
}
return make_pair(d, r);
}

vector<int> chirpz_transform(vector<int> a, int z, int k, int mod) {
    int n = a.size();
    vector<int> x;
    vector<int> y;
    int iz = fpow(z, mod - 2, mod);
    for (int i = 0; i < n; i++) {
        x.push_back((long long)a[i] * fpow(z, (long long)i * i, mod) % mod);
    }
    for (int i = 1 - n; i < k; i++) {
        y.push_back(fpow(iz, (long long)i * i, mod));
    }
    vector<int> r = FFT::multiply(x, y, mod);
    vector<int> res(k);
    for (int i = 0; i < k; i++) {
        res[i] = (long long)r[i + n - 1] * fpow(z, (long long)i * i, mod) %
mod;
    }
    return res;
}

} // namespace FFT

```

27 NTT

```

const int MODULO = 998244353;
const int ROOT = 3; // Primitive root
void fft(vector<int> &a, bool invert) {
    int n = a.size();
    assert((n & (n - 1)) == 0);
    int lg = __builtin_ctz(n);
    for (int i = 0; i < n; ++i) {
        int j = 0;

```

```

    for (int k = 0; k < lg; ++k) if ((i&1<<k)!=0) j |= 1 <<
        (lg-k-1);
    if (i < j) swap(a[i], a[j]);
}
for (int len = 2; len <= n; len *= 2) {
    int wlen = power(ROOT, (MODULO - 1) / len);
    if (invert) wlen = inverse(wlen);
    for (int i = 0; i < n; i += len) {
        int w = 1;
        for (int j = 0; j < len / 2; ++j) {
            int u = a[i + j];
            int v = 1LL * a[i + j + len / 2] * w % MODULO;
            a[i + j] = (u + v) % MODULO;
            a[i + j + len / 2] = (u - v + MODULO) % MODULO;
            w = 1LL * w * wlen % MODULO;
        }
    }
}
if (invert) {
    int mul = inverse(n);
    for (auto &x : a) x = 1LL * x * mul % MODULO;
}

998244353 = 119 * 2^23 + 1. Primitive root: 3.
985661441 = 235 * 2^22 + 1. Primitive root: 3.
1012924417 = 483 * 2^21 + 1. Primitive root: 5

```

28 Gauss

```

// INPUT:      a[][] = an n×n matrix
//             b[][] = an n×m matrix
// OUTPUT:     X      = an n×m matrix (stored in b[][])
//             A^{-1} = an n×n matrix (stored in a[][])
//             returns determinant of a[][]

const double EPS = 1e-10;
typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
T GaussJordan(VVT &a, VVT &b) {
    const int n = a.size();
    const int m = b[0].size();
    VI irow(n), icol(n), ipiv(n);

```

```

    T det = 1;
    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j;
                    pk = k; }
        if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl;
            exit(0); }
        ipiv[pk]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        if (pj != pk) det *= -1;
        irow[i] = pj;
        icol[i] = pk;
        T c = 1.0 / a[pk][pk];
        det *= a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
        }
    }
    for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
        for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
    }
    return det;
}

```

29 Simplex

```

// Two-phase simplex algorithm for solving linear programs of the form
//
//      maximize      c^T x
//      subject to    Ax <= b
//                   x >= 0
//
// INPUT: A -- an m x n matrix

```

```
//      b -- an m-dimensional vector
//      c -- an n-dimensional vector
//      x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments. Then, call Solve(x).

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
    int m, n;
    VI B, N;
    VVD D;

    LPSolver(const VVD &A, const VD &b, const VD &c) :
        m(b.size()), n(c.size()), B(m), N(n + 1), D(m + 2, VD(n + 2)) {
        for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j]
            = A[i][j];
        for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i]
            [n + 1] = b[i]; }
        for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m + 1][n] = 1;
    }

    void Pivot(int r, int s) {
        for (int i = 0; i < m + 2; i++) if (i != r)
            for (int j = 0; j < n + 2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] / D[r][s];
        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] /= D[r][s];
        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] /= -D[r][s];
        D[r][s] = 1.0 / D[r][s];
        swap(B[r], N[s]);
    }

    bool Simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
```

```
while (true) {
    int s = -1;
    for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || (D[x][j] == D[x][s] &&
            N[j] < N[s])) s = j;
    }
    if (D[x][s] > -EPS) return true;
    int r = -1;
    for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][
            s] || ((D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][
            s]) && B[i] < B[r])) r = i;
    }
    if (r == -1) return false;
    Pivot(r, s);
}

DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
    if (D[r][n + 1] < -EPS) {
        Pivot(r, n);
        if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -
            numeric_limits<DOUBLE>::infinity();
        for (int i = 0; i < m; i++) if (B[i] == -1) {
            int s = -1;
            for (int j = 0; j <= n; j++)
                if (s == -1 || D[i][j] < D[i][s] || (D[i][j] == D[i][s]
                    && N[j] < N[s])) s = j;
            Pivot(i, s);
        }
    }
    if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
    return D[m][n + 1];
}

};

int main() {
```

```

const int m = 4;
const int n = 3;
DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
};

DOUBLE _b[m] = { 10, -4, 5, -5 };
DOUBLE _c[n] = { 1, -1, 0 };

VVD A(m);
VD b(_b, _b + m);
VD c(_c, _c + n);
for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

LPSolver solver(A, b, c);
VD x;
DOUBLE value = solver.Solve(x);

cerr << "VALUE: " << value << endl; // VALUE: 1.29032
cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
cerr << endl;
return 0;
}

```

30 Chinese Remainder

```

// Solve linear congruences equation:
// - a[i] * x = b[i] MOD m[i] (mi don't need to be co-prime)
// Tested:
// - https://open.kattis.com/problems/generalchineseremainder
bool linearCongruences(const vector<ll> &a, const vector<ll> &b,
    const vector<ll> &m, ll &x, ll &M) {
    ll n = a.size();
    x = 0; M = 1;
    REP(i, n) {
        ll a_ = a[i] * M, b_ = b[i] - a[i] * x, m_ = m[i];
        ll y, t, g = extgcd(a_, m_, y, t);
        if (b_ % g) return false;
        b_ /= g; m_ /= g;
        x += M * (y * b_ % m_);
    }
}

```

```

        M *= m_;
    }
    x = (x + M) % M;
    return true;
}

```

31 Primitive Root

```

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;
}

```

32 Range Prime Counting

```

// Primes up to 10^12 can be counted in ~1 second.
const int MAXN = 1000005; // MAXN is the maximum value of sqrt(N) + 2
bool prime[MAXN];
int prec[MAXN];
vector<int> P;
void init() {
    prime[2] = true;
    for (int i = 3; i < MAXN; i += 2) prime[i] = true;
    for (int i = 3; i*i < MAXN; i += 2) {
        if (prime[i]) {
            for (int j = i*i; j < MAXN; j += i) prime[j] = false;
        }
    }
    for (int i=1; i<MAXN; i++) {

```

```

        if (prime[i]) P.push_back(i);
        prec[i] = prec[i-1] + prime[i];
    }
}

lint rec(lint N, int K) {
    if (N <= 1 || K < 0) return 0;
    if (N <= P[K]) return N-1;
    if (N < MAXN && 1ll * P[K]*P[K] > N) return N-1 - prec[N] +
        prec[P[K]];
    const int LIM = 250;
    static int memo[LIM*LIM][LIM];
    bool ok = N < LIM*LIM;
    if (ok && memo[N][K]) return memo[N][K];
    lint ret = N/P[K] - rec(N/P[K], K-1) + rec(N, K-1);
    if (ok) memo[N][K] = ret;
    return ret;
}

lint count_primes(lint N) { //less than or equal to
    if (N < MAXN) return prec[N];
    int K = prec[(int)sqrt(N) + 1];
    return N-1 - rec(N, K) + prec[P[K]];
}

```

33 Knight's shortest path

```

int KSP(int x,int 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);
}

```

34 Extended Euclid

Gia su ket qua la (x_0, y_0) , ho nghiem la $(x_0 + k * b / d, y_0 - k * a / d)$

Phuong trinh $ax + by = d$ co nghiem khi va chi khi d chia het cho $\gcd(a, b)$

$a x + b y = \gcd(a, b)$

```

int extgcd(int a, int b, int &x, int &y) {

```

```

    int g = a; x = 1; y = 0;
    if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
    return g;
}

```

35 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;
}

```

36 Sqrt Mod

```

// Jacobi Symbol (m/n), m,n >= 0 and n is odd
#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:  $n(p-1)/2 = -1 \pmod p$ 
int sqrtMod(int n, int p) { //find x:  $x^2 = n \pmod p$  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;
}

```

37 Interval line

```

class IntervalLineTree {
private:
    int n;
    Node *root;
    void update(Node *node, int lo, int hi, Line &line) {
        int mid = (lo + hi) >> 1;
        if (line.getY(lo) <= line.getY(hi) && line.getY(hi) <= node->line.getY(hi))
            return;

        if (line.getY(lo) >= node->line.getY(lo) &&
            line.getY(hi) >= node->line.getY(hi)) {
            node->line = line;
            return;
        }
        // Todo: add left and right note
        if (line.getY(lo) <= node->line.getY(lo) &&
            line.getY(mid) <= node->line.getY(mid)) {
            update(node->rightNode, mid + 1, hi, line);
            return;
        }
        if (line.getY(lo) >= node->line.getY(lo) &&
            line.getY(mid) >= node->line.getY(mid)) {
            update(node->rightNode, mid + 1, hi, node->line);
            node->line = line;
            return;
        }
        if (line.getY(mid+1) <= node->line.getY(mid+1) &&
            line.getY(hi) <= node->line.getY(hi)) {

```

```

            update(node->leftNode, lo, mid, line);
        }
    }
    if (line.getY(mid + 1) >= node->line.getY(mid + 1) &&
        line.getY(hi) >= node->line.getY(hi)) {
        update(node->leftNode, lo, mid, node->line);
        node->line = line;
    }
}
long long get(Node *node, int lo, int hi, int pos) {
    if (lo > pos || hi < pos) return 0;
    long long res = node->line.getY(pos);
    if (lo == hi) return res;
    int mid = (lo + hi) >> 1;
    if (node->leftNode != NULL)
        res = max(res, get(node->leftNode, lo, mid, pos));
    if (node->rightNode != NULL) {
        res = max(res, get(node->rightNode, mid + 1, hi, pos));
    }
    return res;
}
public:
    IntervalLineTree(int _n) {
        n = _n;
        root = new Node();
    }
    void update(Line &line) { update(root, 1, n, line); }
    long long get(int pos) { return get(root, 1, n, pos); }
};

```

38 BIT 2D

```

class BIT2D {
public:
    vector<int> nodes[maxn];
    vector<int> f[maxn];
    void fakeUpdate(int u, int v) {
        for (int x = u; x <= n; x += x & -x)
            nodes[x].push_back(v);
    }
    void update(int u, int v) {
        for (int x = u; x <= n; x += x & -x)
            for (int y = lower_bound(nodes[x].begin(), nodes[x].end(), v) -
                nodes[x].begin() + 1; y <= nodes[x].size();

```

```

        y += y & -y)
        f[x][y]++;
    }
    int get(int u, int v) {
        int res = 0;
        for (int x = u; x > 0; x -= x & -x)
            for (int y = upper_bound(nodes[x].begin(), nodes[x].end(), v) -
                nodes[x].begin(); y > 0; y -= y & -y)
                res += f[x][y];
        return res;
    }
    void prepare(vector<pair<int, int>> queries) {
        reverse(queries.begin(), queries.end());
        for (auto query : queries) {
            fakeUpdate(query.first, query.second);
        }
        reverse(queries.begin(), queries.end());
        for (int i = 1; i <= n; i++) {
            nodes[i].push_back(0);
            sort(nodes[i].begin(), nodes[i].end());
            f[i].resize(((int) nodes[i].size()) + 3);
        }
    }
} bit2D;

```

39 Heavy-Light Decomposition

```

void hld(int u) {
    //Neu chuoì hien tai chua co dinh dau dinh gan goc nhat thi dat u lam
    //dinh dau cua no
    if (chainHead[nChain] == 0) chainHead[nChain] = u;
    //Gan chuoì hien tai cho u
    chainInd[u] = nChain;
    //Giai thich ben duoi
    posInBase[u] = ++nBase;
    // Bien luu dinh con dac biet
    int mxVtx = -1;
    // Tim dinh con dac biet trong so nhung dinh con cua u
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (v != parent[u]) {
            if (mxVtx == -1 || nChild[v] > nChild[mxVtx]) {
                mxVtx = v;
            }
        }
    }
}

```

```

    }
}
//Neu tim ra dinh con dac biet (u khong phai la dinh la) thi di chuyen
//den dinh do
if (mxVtx > -1)
    hld(mxVtx);
// Sau khi di het mot chuoì thi tang nChain len va bat dau mot chuoì moi
for (int i = 0; i < adj[u].size(); i++) {
    int v = adj[u][i];
    if (v != parent[u] && v != mxVtx) {
        nChain++;
        hld(v);
    }
}
}
void update(int u, int a) {
    // uchain chuoì hien tai cua u
    // achain chuoì hien tai cua a
    int uchain = chainInd[u], achain = chainInd[a];
    while (1) {
        // Neu u va a cung nam tren mot chuoì thi update doan tu u den a va
        // ket thuc
        if (uchain == achain) {
            updateIntervalTree(..., posInBase[a], posInBase[u], ...);
            break;
        }
        // Neu u va a khong nam tren cung mot chuoì thi update doan tu u den
        //dinh dau cua chuoì hien tai
        updateIntervalTree(..., posInBase[chainHead[uchain]], posInBase[u],
            ...);
        // Nhay len dinh cha cua dinh dau hien tai
        u = parent[chainHead[uchain]];
        uchain = chainInd[u];
    }
}

```


$$\pi(x) = \lfloor x \rfloor - \sum_{i=1}^a \left\lfloor \frac{x}{p_i} \right\rfloor + \sum_{1 \leq i \leq j \leq a} \left\lfloor \frac{x}{p_i p_j} \right\rfloor - \dots + \frac{1}{2}(b+a-2)(b-a+1) - \sum_{a < i \leq b} \pi\left(\frac{x}{p_i}\right) - \sum_{i=a+1}^c \sum_{j=i}^{b_i} \left[\pi\left(\frac{x}{p_i p_j}\right) - (j-1) \right], a = \pi(x^{1/4}), b = \pi(x^{1/2}), b_i = \pi(\sqrt{x/p_i}), c = \pi(x^{1/3})$$

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

$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440$
 Number of permutations of length n with k cycles:

$$s(n+1, k) = ns(n, k) + s(n, k-1)$$

Number of ways to partition a set of n labelled objects into k nonempty subsets:

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n = kS(n-1, k) + S(n, k-1)$$

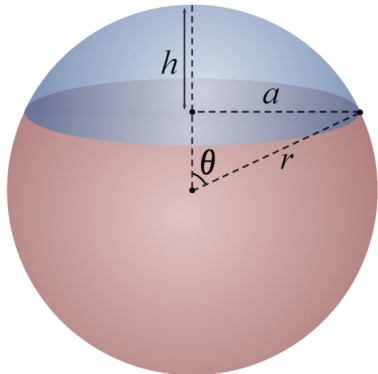
$$H_n = \sum_{k=1}^n \frac{1}{k} \approx \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \frac{1}{252n^6} + \dots$$

$$\frac{1}{2(n+1)} < H_n - \ln n - \gamma < \frac{1}{2n}; \frac{1}{24(n+1)^2} < H_n - \ln\left(n + \frac{1}{2}\right) - \gamma < \frac{1}{24n^2}$$

$$\gamma = 0.57721566490153286060651209008240243104215933593992$$

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

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



Maximum Flows with Edge Demands: $c'(s' \rightarrow v) = \sum_{u \in V} d(u \rightarrow v), c'(v \rightarrow t') =$

$\sum_{w \in V} d(v \rightarrow w), c'(u \rightarrow v) = c(u \rightarrow v) - d(u \rightarrow v), c'(t \rightarrow s) = \infty$. **If feasible:**
 $c_f(u \rightarrow v) = c(u \rightarrow v) - f(u \rightarrow v)$ if $u \rightarrow v \in E$; $f(v \rightarrow u) - d(v \rightarrow u)$ if $v \rightarrow u \in E$, 0 otherwise.

$$\sum_{i=0}^r \binom{n+i}{n} = \binom{n+r+1}{r}$$