Contents

1 Flow

1.1 MinCostMaxFlow

```
struct MincostMaxflow {
    struct Edge {
        int to, rev, cap, w;
        Edge() {}
        Edge(int a, int b, int c, int d): to(a), cap(b)
            , w(c), rev(d) {}
    };
    int n, s, t, p[maxn], id[maxn];
    int d[maxn];
    bool inque[maxn];
    vector<Edge> G[maxn];
    pair<int, int> spfa() {
        memset(p, -1, sizeof(-1));
        fill(d, d + maxn, inf);
        memset(id, -1, sizeof(id));
        d[s] = 0; p[s] = s;
        queue<int> que; que.push(s); inque[s] = true;
        while (que.size()) {
            int tmp = que.front(); que.pop();
            inque[tmp] = false;
            int i = 0;
            for (auto e : G[tmp]) {
                if (e.cap > 0 && d[e.to] > d[tmp] + e.w
                    ) {
                    d[e.to] = d[tmp] + e.w;
                    p[e.to] = tmp;
                    id[e.to] = i;
                    if (!inque[e.to]) que.push(e.to),
                        inque[e.to] = true;
                }
                ++i;
            }
        }
        if (d[t] == inf) return make_pair(-1, -1);
        int a = inf;
        for (int i = t; i != s; i = p[i]) {
            a = min(a, G[p[i]][id[i]].cap);
        }
        for (int i = t; i != s; i = p[i]) {
            Edge &e = G[p[i]][id[i]];
            e.cap -= a; G[e.to][e.rev].cap += a;
        }
        return make_pair(a, d[t]);
    }
    MincostMaxflow(int _n, int _s, int _t): n(_n), s(_s
        ), t(_t) {
        fill(G, G + maxn, vector<Edge>());
    }
    void add_edge(int a, int b, int cap, int w) {
        G[a].push_back(Edge(b, cap, w, (int)G[b].size()
            ));
        G[b].push_back(Edge(a, 0, -w, (int)G[a].size()
            - 1));
    }
    pair<int, int> maxflow() {
        int mxf = 0, mnc = 0;
        while (true) {
            pair<int, int> res = spfa();
            if (res.first == -1) break;
            mxf += res.first; mnc += res.first * res.
                second;
        }
        return make_pair(mxf, mnc);
    }
};
```

2 Graph

2.1 Heavy-Light Decomposition

```
struct HeavyLightDecomp {
    vector<int> G[maxn];
```

```
    int tin[maxn], top[maxn], dep[maxn], maxson[maxn],
        sz[maxn], p[maxn], n, clk;
    void dfs(int now, int fa, int d) {
        dep[now] = d;
        maxson[now] = -1;
        sz[now] = 1;
        p[now] = fa;
        for (int u : G[now]) if (u != fa) {
            dfs(u, now, d + 1);
            sz[now] += sz[u];
            if (maxson[now] == -1 || sz[u] > sz[maxson[
                now]]) maxson[now] = u;
        }
    }
    void link(int now, int t) {
        top[now] = t;
        tin[now] = ++clk;
        if (maxson[now] == -1) return;
        link(maxson[now], t);
        for (int u : G[now]) if (u != p[now]) {
            if (u == maxson[now]) continue;
            link(u, u);
        }
    }
    HeavyLightDecomp(int n): n(n) {
        clk = 0;
        memset(tin, 0, sizeof(tin)); memset(top, 0,
            sizeof(top)); memset(dep, 0, sizeof(dep));
        memset(maxson, 0, sizeof(maxson)); memset(sz,
            0, sizeof(sz)); memset(p, 0, sizeof(p));
    }
    void add_edge(int a, int b) {
        G[a].push_back(b);
        G[b].push_back(a);
    }
    void solve() {
        dfs(0, -1, 0);
        link(0, 0);
    }
    int lca(int a, int b) {
        int ta = top[a], tb = top[b];
        while (ta != tb) {
            if (dep[ta] < dep[tb]) {
                swap(ta, tb); swap(a, b);
            }
            a = p[ta]; ta = top[a];
        }
        if (a == b) return a;
        return dep[a] < dep[b] ? a : b;
    }
    vector<pair<int, int>> get_path(int a, int b) {
        int ta = top[a], tb = top[b];
        vector<pair<int, int>> ret;
        while (ta != tb) {
            if (dep[ta] < dep[tb]) {
                swap(ta, tb); swap(a, b);
            }
            ret.push_back(make_pair(tin[ta], tin[a]));
            a = p[ta]; ta = top[a];
        }
        ret.push_back(make_pair(min(tin[a], tin[b]),
            max(tin[a], tin[b])));
        return ret;
    }
};
```

3 Math

3.1 FFT

```
void prefft() {
    for (int i = 0; i <= maxn; ++i) omega[i] = exp(i *
        2 * pi / maxn * I);
}

void fft(vector<complex<double>>& a, int n, bool inv=
    false) {
    int basic = maxn / n;
    int theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
        int h = m >> 1;
        for (int i = 0; i < h; ++i) {
            complex<double> w = omega[inv ? maxn - (i *
                theta % maxn) : i * theta % maxn];
            for (int j = i; j < n; j += m) {
                int k = j + h;
                complex<double> x = a[j] - a[k];
                a[j] += a[k];
                a[k] = w * x;
            }
        }
        theta = (theta * 2) % maxn;
    }
    int i = 0;
    for (int j = 1; j < n - 1; ++j) {
        for (int k = n >> 1; k > (i ^= k); k >>= 1);
        if (j < i) swap(a[i], a[j]);
    }
    if (inv) for (int i = 0; i < n; ++i) a[i] /= (
        double)n;
}

void invfft(vector<complex<double>>& a, int n) {
    fft(a, n, true);
}
```

3.2 Miller Rabin

```
// n < 4759123141      chk = [2, 7, 61]
// n < 1122004669633   chk = [2, 13, 23, 1662803]
// n < 2^64            chk = [2, 325, 9375, 28178,
    450775, 9780504, 1795265022]

long long fpow(long long a, long long n, long long mod)
    {
    long long ret = 1LL;
    for (; n; n >>= 1) {
        if (n & 1) ret = (ret * a) % mod;
        a = (a * a) % mod;
    }
    return ret;
}

bool check(long long a, long long u, long long n, int t
    ) {
    a = fpow(a, u, n);
    if (a == 0) return true;
    if (a == 1 || a == n - 1) return true;
    for (int i = 0; i < t + 1; ++i) {
        a = a * a % n;
        if (a == 1) return false;
        if (a == n - 1) return true;
    }
    return false;
}

bool miller_rabin(long long n) {
    if (n < 2) return false;
    if (n % 2 == 0) return n == 2;
    long long u = n - 1; int t = 0;
    for (; u & 1; u >>= 1, ++t);
    for (long long i : chk) {
        if (!check(i, u, n, t)) return false;
    }
    return true;
}
```