DinicWithScaling

```
#define pb push_back

struct Dinic{
    struct edge{
        int to, flow, cap;
    };

    const static int N = 555; //count of vertices

    vector<edge> e;
    vector<int> g[N + 7];
    int dp[N + 7];
    int ptr[N + 7];

    void clear(){
        for (int i = 0; i < N + 7; i++) g[i].clear();
        e.clear();
    }

    void addEdge(int a, int b, int cap){
        g[a].pb(e.size());
        e.pb({b, 0, cap});
        g[b].pb(e.size());
        e.pb({a, 0, 0});
    }

    int minFlow, start, finish;

    bool bfs(){
        for (int i = 0; i < N; i++) dp[i] = -1;
        dp[start] = 0;
        vector<int> st;
        int uk = 0;
        st.pb(start);
        while(uk < st.size()){
            int v = st[uk++];
            for (int to : g[v]){
                auto ed = e[to];
                if (ed.cap - ed.flow >= minFlow && dp[ed.to] == -1){
                    dp[ed.to] = dp[v] + 1;
                    st.pb(ed.to);
                }
            }
        }
        return dp[finish] != -1;
    }

    int dfs(int v, int flow){
        if (v == finish) return flow;
        for (; ptr[v] < g[v].size(); ptr[v]++){
            int to = g[v][ptr[v]];
            edge ed = e[to];
            if (ed.cap - ed.flow >= minFlow && dp[ed.to] == dp[v] + 1){
                int add = dfs(ed.to, min(flow, ed.cap - ed.flow));
                if (add){
                    e[to].flow += add;
                    e[to ^ 1].flow -= add;
                    return add;
                }
            }
        }
        return 0;
    }

    int dinic(int start, int finish){
        Dinic::start = start;
```

```
        Dinic::finish = finish;
        int flow = 0;
        for (minFlow = (1 << 30); minFlow; minFlow >>= 1){
            while(bfs()){
                for (int i = 0; i < N; i++) ptr[i] = 0;
                while(int now = dfs(start, (int)2e9 + 7)) flow += now;
            }
        }
        return flow;
    }
} dinic;
```

FFT

```
#define db long double

class cn{
public:
        db x, y;
        cn(){}
        cn(db xx, db yy): x(xx), y(yy) {}
        cn(db xx): x(xx), y(0) {}
        db real() { return x; }
        void operator /= (double f) { x /= f; y /= f; }
};

cn operator + (cn a, cn b) { return cn(a.x + b.x, a.y + b.y); }
cn operator - (cn a, cn b) { return cn(a.x - b.x, a.y - b.y); }
cn operator * (cn a, cn b) { return cn(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
}

class FFT{
public:
        constexpr const static db pi = acos(-1.0);
        const static int MAX_SIZE = 1 << 21;

        //#define cn complex<db>

        int n;
        cn a[MAX_SIZE * 2 + 7], b[MAX_SIZE * 2 + 7];

        int getReverse(int a, int k){
                int ans = 0;
                for (int i = 0; i < k; i++) if ((a >> i) & 1) ans ^= (1 << (k - i -
1));
                return ans;
        }

        void fft(cn *a, int type){
                int k = -1;
                for (int i = 0; i < 25; i++) if ((n >> i) & 1){
                        k = i;
                        break;
                }
                for (int i = 0; i < n; i++){
                        int j = getReverse(i, k);
                        if (i < j) swap(a[i], a[j]);
                }
                for (int len = 2; len <= n; len *= 2){
                        cn w(cos(2 * pi / (db)len), sin(2 * pi / (db)len) * type);
                        for (int i = 0; i < n; i += len){
                                cn g = cn(1, 0);
                                for (int j = 0; j < len / 2; j++){
                                        cn x = a[i + j];
                                        cn y = a[i + j + len / 2] * g;
                                        a[i + j] = x + y;
                                        a[i + j + len / 2] = x - y;
                                        g = g * w;
                                }
```

```
                        }
                }
                if (type == -1) for (int i = 0; i < n; i++) a[i] /= n;
        }

        vector<int> mult(vector<int> &w1, vector<int> &w2){
                n = 1;
                while(n < w1.size() + w2.size()) n *= 2;
                for (int i = 0; i < w1.size(); i++) a[i] = w1[i];
                for (int i = 0; i < w2.size(); i++) b[i] = w2[i];
                for (int i = w1.size(); i < n; i++) a[i] = 0;
                for (int i = w2.size(); i < n; i++) b[i] = 0;
                fft(a, 1);
                fft(b, 1);
                for (int i = 0; i < n; i++) a[i] = a[i] * b[i];
                fft(a, -1);
                vector<int> ans(n);
                for (int i = 0; i < n; i++) ans[i] = floor((db)a[i].real()
                 + 0.5);
                while(ans.size() && ans.back() == 0) ans.pop_back();
                return ans;
        }
};
```

FlowCirculation

```
#define pb push_back

struct Dinic{
    struct edge{
        int to, flow, cap;
    };

    const static int N = 555; //count of vertices

    vector<edge> e;
    vector<int> g[N + 7];
    int dp[N + 7];
    int ptr[N + 7];

    void clear(){
        for (int i = 0; i < N + 7; i++) g[i].clear();
        e.clear();
    }

    void addEdge(int a, int b, int cap){
                g[a].pb(e.size());
                e.pb({b, 0, cap});
        g[b].pb(e.size());
        e.pb({a, 0, 0});
    }

    void addCircular(int a, int b, int l, int r) {
        addEdge(S, b, l); //S - source
        addEdge(a, T, l); //T - sink
        addEdge(a, b, r - l);
    }

    int minFlow, start, finish;

    bool bfs(){
        for (int i = 0; i < N; i++) dp[i] = -1;
        dp[start] = 0;
        vector<int> st;
        int uk = 0;
        st.pb(start);
        while(uk < st.size()){
            int v = st[uk++];
            for (int to : g[v]){
```

```
                auto ed = e[to];
                if (ed.cap - ed.flow >= minFlow && dp[ed.to] == -1){
                    dp[ed.to] = dp[v] + 1;
                    st.pb(ed.to);
                }
            }
        }
        return dp[finish] != -1;
    }

    int dfs(int v, int flow){
        if (v == finish) return flow;
        for (; ptr[v] < g[v].size(); ptr[v]++){
            int to = g[v][ptr[v]];
            edge ed = e[to];
            if (ed.cap - ed.flow >= minFlow && dp[ed.to] == dp[v] + 1){
                int add = dfs(ed.to, min(flow, ed.cap - ed.flow));
                if (add){
                    e[to].flow += add;
                    e[to ^ 1].flow -= add;
                    return add;
                }
            }
        }
        return 0;
    }

    int dinic(int start, int finish){
        Dinic::start = start;
        Dinic::finish = finish;
        int flow = 0;
        for (minFlow = (1 << 30); minFlow; minFlow >>= 1){
            while(bfs()){
                for (int i = 0; i < N; i++) ptr[i] = 0;
                while(int now = dfs(start, (int)2e9 + 7)) flow += now;
            }
        }
        return flow;
    }
} dinic;
```

NTT

```
class NTT{
public:
        #define db long double
        #define ll long long
        const static int mod = 998244353;
        const static int root = 646; // 646^(2^20) == 1 (998244353)
        const static int rev_root = 208611436;
        const static int MAX_SIZE = 1 << 21;

        void add(int &a, int b){
                a += b;
                if (a < 0) a += mod;
                if (a >= mod) a -= mod;
        }

        int sum(int a, int b){
                add(a, b);
                return a;
        }

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

        int bp(int a, int k){
                if (k == 0) return 1;
```

```
                if (k & 1){
                        return mult(a, bp(a, k - 1));
                } else {
                        int q = bp(a, k >> 1);
                        return mult(q, q);
                }
        }

        int rev(int a){
                return bp(a, mod - 2);
        }

        int n;
        int a[MAX_SIZE * 2 + 7], b[MAX_SIZE * 2 + 7];

        int getReverse(int a, int k){
                int ans = 0;
                for (int i = 0; i < k; i++) if ((a >> i) & 1) ans ^= (1 << (k - i -
1));
                return ans;
        }

        void ntt(int *a, int type){
                int k = -1;
                for (int i = 0; i < 25; i++) if ((n >> i) & 1){
                        k = i;
                        break;
                }
                for (int i = 0; i < n; i++){
                        int j = getReverse(i, k);
                        if (i < j) swap(a[i], a[j]);
                }
                for (int len = 2; len <= n; len *= 2){
                        int w = bp(root, (1 << 20) / len);
                        if (type == -1) w = bp(rev_root, (1 << 20) / len);
                        for (int i = 0; i < n; i += len){
                                int g = 1;
                                for (int j = 0; j < len / 2; j++){
                                        int x = a[i + j];
                                        int y = mult(a[i + j + len / 2], g);
                                        a[i + j] = sum(x, y);
                                        a[i + j + len / 2] = sum(x, mod - y);
                                        g = mult(g, w);
                                }
                        }
                }
                if (type == -1){
                        int rev_n = rev(n);
                        for (int i = 0; i < n; i++) a[i] = mult(a[i], rev_n);
                }
        }

        vector<int> mult(vector<int> &w1, vector<int> &w2){
                n = 1;
                while(n < w1.size() + w2.size()) n *= 2;
                for (int i = 0; i < w1.size(); i++){
                        a[i] = w1[i];
                        a[i] %= mod;
                        if (a[i] < 0) a[i] += mod;
                }
                for (int i = 0; i < w2.size(); i++){
                        b[i] = w2[i];
                        b[i] %= mod;
                        if (b[i] < 0) b[i] += mod;
                }
                for (int i = w1.size(); i < n; i++) a[i] = 0;
                for (int i = w2.size(); i < n; i++) b[i] = 0;
                ntt(a, 1);
                ntt(b, 1);
                for (int i = 0; i < n; i++) a[i] = mult(a[i], b[i]);
                ntt(a, -1);
```

```
                vector<int> ans(n);
                for (int i = 0; i < n; i++) ans[i] = a[i];
                while(ans.size() && ans.back() == 0) ans.pop_back();
                return ans;
        }
};
```