Grimoire's Standard Code Library[*]

Shanghai Jiao Tong University

Dated: 2017 年 9 月 24 日

[*]https://github.com/kzoacn/Grimoire

目录

Chapter 1

代数

$O(n^2 \log n)$ 求线性递推数列第 n 项

Given $a_0, a_1, \cdots, a_{m-1}$
$a_n = c_0 * a_{n-m} + \cdots + c_{m-1} * a_0$
a_0 *is the nth element*, $\cdots$, a_{m-1} *is the* $n+m-1$*th element*

```cpp
void linear_recurrence(long long n, int m, int a[], int c[], int p) {
    long long v[M] = {1 % p}, u[M << 1], msk = !!n;
    for(long long i(n); i > 1; i >>= 1) {
        msk <<= 1;
    }
    for(long long x(0); msk; msk >>= 1, x <<= 1) {
        fill_n(u, m << 1, 0);
        int b(!!(n & msk));
        x |= b;
        if(x < m) {
            u[x] = 1 % p;
        }else {
            for(int i(0); i < m; i++) {
                for(int j(0), t(i + b); j < m; j++, t++) {
                    u[t] = (u[t] + v[i] * v[j]) % p;
                }
            }
            for(int i((m << 1) - 1); i >= m; i--) {
                for(int j(0), t(i - m); j < m; j++, t++) {
                    u[t] = (u[t] + c[j] * u[i]) % p;
                }
            }
        }
        copy(u, u + m, v);
    }
    //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
    for(int i(m); i < 2 * m; i++) {
        a[i] = 0;
        for(int j(0); j < m; j++) {
            a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
        }
    }
    for(int j(0); j < m; j++) {
        b[j] = 0;
        for(int i(0); i < m; i++) {
            b[j] = (b[j] + v[i] * a[i + j]) % p;
        }
    }
    for(int j(0); j < m; j++) {
        a[j] = b[j];
    }
}
```

闪电数论变换与魔力 CRT

```
#define meminit(A, l, r) memset(A + (l), 0, sizeof(*A) * ((r) - (l)))
#define memcopy(B, A, l, r) memcpy(B, A + (l), sizeof(*A) * ((r) - (l)))
void DFT(int *a, int n, int f) { //f=1 逆 DFT
    for (register int i = 0, j = 0; i < n; i++) {
        if (i > j) std::swap(a[i], a[j]);
        for (register int t = n >> 1; (j ^= t) < t; t >>= 1);
    }
    for (register int i = 2; i <= n; i <<= 1) {
        static int exp[MAXN];
        exp[0] = 1; exp[1] = fpm(PRT, (MOD - 1) / i, MOD);
        if (f == 1) exp[1] = fpm(exp[1], MOD - 2, MOD);
        for (register int k = 2; k < (i >> 1); k++) {
            exp[k] = 1ll * exp[k - 1] * exp[1] % MOD;
        }
        for (register int j = 0; j < n; j += i) {
            for (register int k = 0; k < (i >> 1); k++) {
                register int &pA = a[j + k], &pB = a[j + k + (i >> 1)];
                register long long B = 1ll * pB * exp[k];
                pB = (pA - B) % MOD;
                pA = (pA + B) % MOD;
            }
        }
    }
    if (f == 1) {
        register int rev = fpm(n, MOD - 2, MOD);
        for (register int i = 0; i < n; i++) {
            a[i] = 1ll * a[i] * rev % MOD;
            if (a[i] < 0) { a[i] += MOD; }
        }
    }
}
// 在不写高精度的情况下合并 FFT 所得结果对 MOD 取模后的答案
// 值得注意的是，这个东西不能最后再合并，而是应该每做一次多项式乘法就 CRT 一次
int CRT(int *a) {
    static int x[3];
    for (int i = 0; i < 3; i++) {
        x[i] = a[i];
        for (int j = 0; j < i; j++) {
            int t = (x[i] - x[j] + FFT[i] -> MOD) % FFT[i] -> MOD;
            if (t < 0) t += FFT[i] -> MOD;
            x[i] = 1LL * t * inv[j][i] % FFT[i] -> MOD;
        }
    }
    int sum = 1, ret = x[0] % MOD;
    for (int i = 1; i < 3; i ++) {
        sum = 1LL * sum * FFT[i - 1] -> MOD % MOD;
        ret += 1LL * x[i] * sum % MOD;
        if(ret >= MOD) ret -= MOD;
    }
    return ret;
}
for (int i = 0; i < 3; i++) // inv 数组的预处理过程，inverse(x, p) 表示求 x 在 p 下逆元
    for (int j = 0; j < 3; j++)
        inv[i][j] = inverse(FFT[i] -> MOD, FFT[j] -> MOD);
```

多项式求逆

Given polynomial a and n, b is the polynomial such that $a * b \equiv 1 (\mod x^n)$

```
void getInv(int *a, int *b, int n) {
    static int tmp[MAXN];
    b[0] = fpm(a[0], MOD - 2, MOD);
    for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
        for (; M <= 3 * (c - 1); M <<= 1);
        meminit(b, c, M);
        meminit(tmp, c, M);
        memcopy(tmp, a, 0, c);
        DFT(tmp, M, 0);
        DFT(b, M, 0);
        for (int i = 0; i < M; i++) {
            b[i] = 1ll * b[i] * (2ll - 1ll * tmp[i] * b[i] % MOD + MOD) % MOD;
        }
        DFT(b, M, 1);
        meminit(b, c, M);
    }
}
```

多项式除法

d is quotient and r is remainder

```
void divide(int n, int m, int *a, int *b, int *d, int *r) { // n、m 分别为多项式 A（被除数）
  ↪ 和 B（除数）的指数 + 1
    static int M, tA[MAXN], tB[MAXN], inv[MAXN], tD[MAXN];
    for (; n > 0 && a[n - 1] == 0; n--);
    for (; m > 0 && b[m - 1] == 0; m--);
    for (int i = 0; i < n; i++) tA[i] = a[n - i - 1];
    for (int i = 0; i < m; i++) tB[i] = b[m - i - 1];
    for (M = 1; M <= n - m + 1; M <<= 1);
    if (m < M) meminit(tB, m, M);
    getInv(tB, inv, M);
    for (M = 1; M <= 2 * (n - m + 1); M <<= 1);
    meminit(inv, n - m + 1, M);
    meminit(tA, n - m + 1, M);
    DFT(inv, M, 0);
    DFT(tA, M, 0);
    for (int i = 0; i < M; i++) {
        d[i] = 1ll * inv[i] * tA[i] % MOD;
    }
    DFT(d, M, 1);
    std::reverse(d, d + n - m + 1);
    for (M = 1; M <= n; M <<= 1);
    memcopy(tB, b, 0, m);
    if (m < M) meminit(tB, m, M);
    memcopy(tD, d, 0, n - m + 1);
    meminit(tD, n - m + 1, M);
    DFT(tD, M, 0);
    DFT(tB, M, 0);
    for (int i = 0; i < M; i++) {
        r[i] = 1ll * tD[i] * tB[i] % MOD;
    }
    DFT(r, M, 1);
    meminit(r, n, M);
    for (int i = 0; i < n; i++) {
        r[i] = (a[i] - r[i] + MOD) % MOD;
    }
}
```

多项式取指数取对数

Given polynomial a and n, b is the polynomial such that $b \equiv e^a (\mod x^n)$ or $b \equiv \ln a (\mod x^n)$

```
void getDiff(int *a, int *b, int n) { // 多项式取微分
    for (int i = 0; i + 1 < n; i++) {
        b[i] = 1ll * (i + 1) * a[i + 1] % MOD;
    }
    b[n - 1] = 0;
}
void getInt(int *a, int *b, int n) { // 多项式取积分，积分常数为 0
    static int inv[MAXN];
    inv[1] = 1;
    for (int i = 2; i < n; i++) {
        inv[i] = 1ll * (MOD - MOD / i) * inv[MOD % i] % MOD;
    }
    b[0] = 0;
    for (int i = 1; i < n; i++) {
        b[i] = 1ll * a[i - 1] * inv[i] % MOD;
    }
}
void getLn(int *a, int *b, int n) {
    static int inv[MAXN], d[MAXN];
    int M = 1;
    for (; M <= 2 * (n - 1); M <<= 1);
    getInv(a, inv, n);
    getDiff(a, d, n);
    meminit(d, n, M);
    meminit(inv, n, M);
    DFT(d, M, 0); DFT(inv, M, 0);
    for (int i = 0; i < M; i++) {
        d[i] = 1ll * d[i] * inv[i] % MOD;
    }
    DFT(d, M, 1);
    getInt(d, b, n);
}
void getExp(int *a, int *b, int n) {
    static int ln[MAXN], tmp[MAXN];
    b[0] = 1;
    for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
        for (; M <= 2 * (c - 1); M <<= 1);
        int bound = std::min(c, n);
        memcopy(tmp, a, 0, bound);
        meminit(tmp, bound, M);
        meminit(b, c, M);
        getLn(b, ln, c);
        meminit(ln, c, M);
        DFT(b, M, 0);
        DFT(tmp, M, 0);
        DFT(ln, M, 0);
        for (int i = 0; i < M; i++) {
            b[i] = 1ll * b[i] * (1ll - ln[i] + tmp[i] + MOD) % MOD;
        }
        DFT(b, M, 1);
        meminit(b, c, M);
    }
}
```

Chapter 2

数论

大整数相乘取模

```
// x 与 y 须非负
long long mult(long long x, long long y, long long MODN) {
    long long t = (x * y - (long long)((long double)x / MODN * y + 1e-3) * MODN) % MODN;
    return t < 0 ? t + MODN : t;
}
```

线段下整点

solve for $\sum_{i=0}^{n-1} \lfloor \frac{a+bi}{m} \rfloor$, $n, m, a, b > 0$

```
LL solve(LL n,LL a,LL b,LL m){
    if(b==0) return n*(a/m);
    if(a>=m) return n*(a/m)+solve(n,a%m,b,m);
    if(b>=m) return (n-1)*n/2*(b/m)+solve(n,a,b%m,m);
    return solve((a+b*n)/m,(a+b*n)%m,m,b);
}
```

中国剩余定理

first is remainder, second is module

```
inline void fix(LL &x, LL y) {
    x = (x % y + y) % y;
}
bool solve(int n, std::pair<LL, LL> a[],
                  std::pair<LL, LL> &ans) {
    ans = std::make_pair(1, 1);
    for (int i = 0; i < n; ++i) {
        LL num, y;
        euclid(ans.second, a[i].second, num, y);
        LL divisor = std::__gcd(ans.second, a[i].second);
        if ((a[i].first - ans.first) % divisor) {
            return false;
        }
        num *= (a[i].first - ans.first) / divisor;
        fix(num, a[i].second);
        ans.first += ans.second * num;
        ans.second *= a[i].second / divisor;
        fix(ans.first, ans.second);
    }
    return true;
}
```

Chapter 3

图论

图论基础

```cpp
struct Graph {  // Remember to call .init()!
    int e, nxt[M], v[M], adj[N], n;
    bool base;
    __inline void init(bool _base, int _n = 0) {
        assert(n < N);
        n = _n; base = _base;
        e = 0; memset(adj + base, -1, sizeof(*adj) * n);
    }
    __inline int new_node() {
        adj[n + base] = -1;
        assert(n + base + 1 < N);
        return n++ + base;
    }
    __inline void ins(int u0, int v0) {  // directional
        assert(u0 < n + base && v0 < n + base);
        v[e] = v0; nxt[e] = adj[u0]; adj[u0] = e++;
        assert(e < M);
    }
    __inline void bi_ins(int u0, int v0) {  // bi-directional
        ins(u0, v0); ins(v0, u0);
    }
};
```

闪电二分图匹配

```cpp
int matchx[N], matchy[N], level[N];
vector<int> edge[N];
bool dfs(int x) {
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        int w = matchy[y];
        if (w == -1 || level[x] + 1 == level[w] && dfs(w)) {
            matchx[x] = y;
            matchy[y] = x;
            return true;
        }
    }
    level[x] = -1;
    return false;
}
int solve() {
    memset(matchx, -1, sizeof(*matchx) * n);
    memset(matchy, -1, sizeof(*matchy) * m);
    for (int ans = 0; ; ) {
        std::vector<int> q;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1) {
                level[i] = 0;
                q.push_back(i);
            } else {
                level[i] = -1;
            }
        }
        for (int head = 0; head < (int)q.size(); ++head) {
            int x = q[head];
            for (int i = 0; i < (int)edge[x].size(); ++i) {
                int y = edge[x][i];
                int w = matchy[y];
                if (w != -1 && level[w] < 0) {
                    level[w] = level[x] + 1;
                    q.push_back(w);
                }
            }
        }
        int delta = 0;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1 && dfs(i)) {
                delta++;
            }
        }
        if (delta == 0) {
            return ans;
        } else {
            ans += delta;
        }
    }
```

```
}
```

一般图匹配

```
// 0-base, match[u] is linked to u
vector<int> lnk[MAXN];
int match[MAXN], Queue[MAXN], pred[MAXN], base[MAXN], head, tail, sta, fin, nbase;
bool inQ[MAXN], inB[MAXN];
inline void push(int u) {
    Queue[tail++] = u; inQ[u] = 1;
}
inline int pop() {
    return Queue[head++];
}
inline int FindCA(int u, int v) {
    static bool inP[MAXN];
    fill(inP, inP + n, false);
    while (1) {
        u = base[u]; inP[u] = 1;
        if(u == sta) break;
        u = pred[match[u]];
    }
    while (1) {
        v = base[v];
        if (inP[v]) break;
        v = pred[match[v]];
    }
    return v;
}
inline void RT(int u) {
    int v;
    while (base[u] != nbase) {
        v = match[u];
        inB[base[u]] = inB[base[v]] = 1;
        u = pred[v];
        if (base[u] != nbase) pred[u] = v;
    }
}
inline void BC(int u, int v) {
    nbase = FindCA(u, v);
    fill(inB, inB + n, 0);
    RT(u); RT(v);
    if (base[u] != nbase) pred[u] = v;
    if (base[v] != nbase) pred[v] = u;
    for (int i = 0; i < n; ++i)
```

```
        if (inB[base[i]]) {
            base[i] = nbase;
            if (!inQ[i]) push(i);
        }
}
bool FindAP(int u) {
    bool found = false;
    for (int i = 0; i < n; ++i) {
        pred[i] = -1; base[i] = i; inQ[i] = 0;
    }
    sta = u; fin = -1; head = tail = 0; push(sta);
    while (head < tail) {
        int u = pop();
        for (int i = (int)lnk[u].size() - 1; i >= 0; --i) {
            int v = lnk[u][i];
            if (base[u] != base[v] && match[u] != v) {
                if (v == sta || match[v] >= 0 && pred[match[v]] >= 0) BC(u, v);
                else if (pred[v] == -1) {
                    pred[v] = u;
                    if (match[v] >= 0) push(match[v]);
                    else {
                        fin = v;
                        return true;
                    }
                }
            }
        }
    }
    return found;
}
inline void AP() {
    int u = fin, v, w;
    while (u >= 0) {
        v = pred[u]; w = match[v];
        match[v] = u; match[u] = v;
        u = w;
    }
}
inline int FindMax() {
    for (int i = 0; i < n; ++i) match[i] = -1;
    for (int i = 0; i < n; ++i)
        if (match[i] == -1 && FindAP(i)) AP();
    int ans = 0;
    for (int i = 0; i < n; ++i) {
        ans += (match[i] != -1);
    }
    return ans;
}
```

一般最大权匹配

```
//maximum weight blossom,  change g[u][v].w to INF - g[u][v].w when minimum weight blossom
 ↪ is needed
//type of ans is long long
//replace all int to long long if weight of edge is long long

struct WeightGraph {
    static const int INF = INT_MAX;
    static const int MAXN = 400;
    struct edge{
        int u, v, w;
        edge() {}
        edge(int u, int v, int w): u(u), v(v), w(w) {}
    };
    int n, n_x;
    edge g[MAXN * 2 + 1][MAXN * 2 + 1];
    int lab[MAXN * 2 + 1];
    int match[MAXN * 2 + 1], slack[MAXN * 2 + 1], st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
    int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
    vector<int> flower[MAXN * 2 + 1];
    queue<int> q;
    inline int e_delta(const edge &e){ // does not work inside blossoms
        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
    }
    inline void update_slack(int u, int x){
        if(!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x]))
            slack[x] = u;
    }
    inline void set_slack(int x){
        slack[x] = 0;
        for(int u = 1;u <= n; ++u)
            if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
                update_slack(u, x);
    }
    void q_push(int x){
        if(x <= n)q.push(x);
        else for(size_t i = 0;i < flower[x].size(); i++)
            q_push(flower[x][i]);
    }
    inline void set_st(int x, int b){
        st[x]=b;
        if(x > n) for(size_t i = 0;i < flower[x].size(); ++i)
                    set_st(flower[x][i], b);
    }
    inline int get_pr(int b, int xr){
        int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
        if(pr % 2 == 1){
            reverse(flower[b].begin() + 1, flower[b].end());
            return (int)flower[b].size() - pr;
        } else return pr;
    }
    inline void set_match(int u, int v){
        match[u]=g[u][v].v;
        if(u > n){
            edge e=g[u][v];
            int xr = flower_from[u][e.u], pr=get_pr(u, xr);
            for(int i = 0;i < pr; ++i)
                set_match(flower[u][i], flower[u][i ^ 1]);
            set_match(xr, v);
            rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
        }
    }
    inline void augment(int u, int v){
        for(; ; ){
            int xnv=st[match[u]];
            set_match(u, v);
            if(!xnv)return;
            set_match(xnv, st[pa[xnv]]);
            u=st[pa[xnv]], v=xnv;
        }
    }
    inline int get_lca(int u, int v){
        static int t=0;
        for(++t; u || v; swap(u, v)){
            if(u == 0)continue;
            if(vis[u] == t)return u;
            vis[u] = t;
            u = st[match[u]];
            if(u) u = st[pa[u]];
        }
        return 0;
    }
    inline void add_blossom(int u, int lca, int v){
        int b = n + 1;
        while(b <= n_x && st[b]) ++b;
        if(b > n_x) ++n_x;
        lab[b] = 0, S[b] = 0;
        match[b] = match[lca];
        flower[b].clear();
        flower[b].push_back(lca);
        for(int x = u, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        reverse(flower[b].begin() + 1, flower[b].end());
        for(int x = v, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        set_st(b, b);
        for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
        for(int x = 1; x <= n; ++x) flower_from[b][x] = 0;
        for(size_t i = 0 ; i < flower[b].size(); ++i){
            int xs = flower[b][i];
            for(int x = 1; x <= n_x; ++x)
                if(g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
            for(int x = 1;x <= n; ++x)
                if(flower_from[xs][x]) flower_from[b][x] = xs;
        }
        set_slack(b);
    }
    inline void expand_blossom(int b){ // S[b]  ==  1
        for(size_t i = 0; i < flower[b].size(); ++i)
            set_st(flower[b][i], flower[b][i]);
        int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
        for(int i = 0; i < pr; i += 2){
            int xs = flower[b][i], xns = flower[b][i + 1];
            pa[xs] = g[xns][xs].u;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for(size_t i = pr + 1;i < flower[b].size(); ++i){
            int xs = flower[b][i];
            S[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    inline bool on_found_edge(const edge &e){
        int u = st[e.u], v = st[e.v];
        if(S[v] == -1){
            pa[v] = e.u, S[v] = 1;
            int nu = st[match[v]];
            slack[v] = slack[nu] = 0;
            S[nu] = 0, q_push(nu);
        }else if(S[v] == 0){
            int lca = get_lca(u, v);
            if(!lca) return augment(u, v), augment(v, u), true;
            else add_blossom(u, lca, v);
        }
        return false;
    }
    inline bool matching(){
        memset(S + 1, -1, sizeof(int) * n_x);
        memset(slack + 1, 0, sizeof(int) * n_x);
        q = queue<int>();
        for(int x = 1;x <= n_x; ++x)
            if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
        if(q.empty())return false;
        for(;;){
            while(q.size()){
                int u = q.front();q.pop();
                if(S[st[u]] == 1)continue;
                for(int v = 1;v <= n; ++v)
                    if(g[u][v].w > 0 && st[u] != st[v]){
                        if(e_delta(g[u][v]) == 0){
                            if(on_found_edge(g[u][v]))return true;
                        }else update_slack(u, st[v]);
                    }
            }
            int d = INF;
            for(int b = n + 1; b <= n_x;++b)
                if(st[b] == b && S[b] == 1)d = min(d, lab[b]/2);
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x]){
                    if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
                    else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
                }
            for(int u = 1; u <= n; ++u){
                if(S[st[u]] == 0){
                    if(lab[u] <= d)return 0;
                    lab[u] -= d;
                }else if(S[st[u]] == 1)lab[u] += d;
            }
            for(int b = n+1; b <= n_x; ++b)
```

```
                if(st[b] == b){
                    if(S[st[b]] == 0) lab[b] += d * 2;
                    else if(S[st[b]] == 1) lab[b] -= d * 2;
                }
            q=queue<int>();
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) ==
↪ 0)
                    if(on_found_edge(g[slack[x]][x]))return true;
            for(int b = n + 1; b <= n_x; ++b)
                if(st[b] == b && S[b] == 1 && lab[b] == 0)expand_blossom(b);
        }
        return false;
    }
    inline pair<long long, int> solve(){
        memset(match + 1, 0, sizeof(int) * n);
        n_x = n;
        int n_matches = 0;
        long long tot_weight = 0;
        for(int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();
        int w_max = 0;
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v){
                flower_from[u][v] = (u == v ? u : 0);
                w_max = max(w_max, g[u][v].w);
            }
        for(int u = 1; u <= n; ++u) lab[u] = w_max;
        while(matching()) ++n_matches;
        for(int u = 1; u <= n; ++u)
            if(match[u] && match[u] < u)
                tot_weight += g[u][match[u]].w;
        return make_pair(tot_weight, n_matches);
    }
    inline void init(){
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v)
                g[u][v]=edge(u, v, 0);
    }
};
```

无向图最小割

```
/*
 * Stoer Wagner 全局最小割 O(V ^ 3)
 * 1base, 点数 n, 邻接矩阵 edge[MAXN][MAXN]
```

```cpp
 * 返回值为全局最小割
 */

int StoerWagner() {
    static int v[MAXN], wage[MAXN];
    static bool vis[MAXN];

    for (int i = 1; i <= n; ++i) v[i] = i;

    int res = INF;

    for (int nn = n; nn > 1; --nn) {
        memset(vis, 0, sizeof(bool) * (nn + 1));
        memset(wage, 0, sizeof(int) * (nn + 1));

        int pre, last = 1; // vis[1] = 1;

        for (int i = 1; i < nn; ++i) {
            pre = last; last = 0;
            for (int j = 2; j <= nn; ++j) if (!vis[j]) {
                wage[j] += edge[v[pre]][v[j]];
                if (!last || wage[j] > wage[last]) last = j;
            }
            vis[last] = 1;
        }

        res = std::min(res, wage[last]);

        for (int i = 1; i <= nn; ++i) {
            edge[v[i]][v[pre]] += edge[v[last]][v[i]];
            edge[v[pre]][v[i]] += edge[v[last]][v[i]];
        }
        v[last] = v[nn];
    }
    return res;
}
```

最大带权带花树

```cpp
//maximum weight blossom, change g[u][v].w to INF - g[u][v].w when minimum weight blossom
  ↪ is needed
//type of ans is long long
//replace all int to long long if weight of edge is long long

struct WeightGraph {
    static const int INF = INT_MAX;
    static const int MAXN = 400;
    struct edge{
        int u, v, w;
        edge() {}
        edge(int u, int v, int w): u(u), v(v), w(w) {}
    };
    int n, n_x;
    edge g[MAXN * 2 + 1][MAXN * 2 + 1];
    int lab[MAXN * 2 + 1];
    int match[MAXN * 2 + 1], slack[MAXN * 2 + 1], st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
    int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
    vector<int> flower[MAXN * 2 + 1];
    queue<int> q;
    inline int e_delta(const edge &e){ // does not work inside blossoms
        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
    }
    inline void update_slack(int u, int x){
        if(!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x]))
            slack[x] = u;
    }
    inline void set_slack(int x){
        slack[x] = 0;
        for(int u = 1;u <= n; ++u)
            if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
                update_slack(u, x);
    }
    void q_push(int x){
        if(x <= n)q.push(x);
        else for(size_t i = 0;i < flower[x].size(); i++)
            q_push(flower[x][i]);
    }
    inline void set_st(int x, int b){
        st[x]=b;
        if(x > n) for(size_t i = 0;i < flower[x].size(); ++i)
                    set_st(flower[x][i], b);
    }
    inline int get_pr(int b, int xr){
        int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
        if(pr % 2 == 1){
            reverse(flower[b].begin() + 1, flower[b].end());
            return (int)flower[b].size() - pr;
        } else return pr;
    }
    inline void set_match(int u, int v){
        match[u]=g[u][v].v;
        if(u > n){
            edge e=g[u][v];
            int xr = flower_from[u][e.u], pr=get_pr(u, xr);
            for(int i = 0;i < pr; ++i)
                set_match(flower[u][i], flower[u][i ^ 1]);
            set_match(xr, v);
            rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
        }
    }
    inline void augment(int u, int v){
        for(; ; ){
            int xnv=st[match[u]];
            set_match(u, v);
            if(!xnv)return;
            set_match(xnv, st[pa[xnv]]);
            u=st[pa[xnv]], v=xnv;
        }
    }
    inline int get_lca(int u, int v){
        static int t=0;
        for(++t; u || v; swap(u, v)){
            if(u == 0)continue;
            if(vis[u] == t)return u;
            vis[u] = t;
            u = st[match[u]];
            if(u) u = st[pa[u]];
        }
        return 0;
    }
    inline void add_blossom(int u, int lca, int v){
        int b = n + 1;
        while(b <= n_x && st[b]) ++b;
        if(b > n_x) ++n_x;
        lab[b] = 0, S[b] = 0;
        match[b] = match[lca];
        flower[b].clear();
        flower[b].push_back(lca);
        for(int x = u, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        reverse(flower[b].begin() + 1, flower[b].end());
        for(int x = v, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        set_st(b, b);
        for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
        for(int x = 1; x <= n; ++x) flower_from[b][x] = 0;
        for(size_t i = 0 ; i < flower[b].size(); ++i){
            int xs = flower[b][i];
            for(int x = 1; x <= n_x; ++x)
                if(g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
            for(int x = 1;x <= n; ++x)
                if(flower_from[xs][x]) flower_from[b][x] = xs;
        }
        set_slack(b);
    }
    inline void expand_blossom(int b){ // S[b]  ==  1
        for(size_t i = 0; i < flower[b].size(); ++i)
            set_st(flower[b][i], flower[b][i]);
        int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
        for(int i = 0; i < pr; i += 2){
            int xs = flower[b][i], xns = flower[b][i + 1];
            pa[xs] = g[xns][xs].u;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for(size_t i = pr + 1;i < flower[b].size(); ++i){
            int xs = flower[b][i];
            S[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    inline bool on_found_edge(const edge &e){
        int u = st[e.u], v = st[e.v];
        if(S[v] == -1){
            pa[v] = e.u, S[v] = 1;
            int nu = st[match[v]];
            slack[v] = slack[nu] = 0;
            S[nu] = 0, q_push(nu);
        }else if(S[v] == 0){
            int lca = get_lca(u, v);
            if(!lca) return augment(u, v), augment(v, u), true;
            else add_blossom(u, lca, v);
        }
        return false;
    }
    inline bool matching(){
        memset(S + 1, -1, sizeof(int) * n_x);
        memset(slack + 1, 0, sizeof(int) * n_x);
        q = queue<int>();
        for(int x = 1;x <= n_x; ++x)
            if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
        if(q.empty())return false;
        for(;;){
            while(q.size()){
                int u = q.front();q.pop();
                if(S[st[u]] == 1)continue;
                for(int v = 1;v <= n; ++v)
                    if(g[u][v].w > 0 && st[u] != st[v]){
                        if(e_delta(g[u][v]) == 0){
                            if(on_found_edge(g[u][v]))return true;
                        }else update_slack(u, st[v]);
                    }
            }
            int d = INF;
            for(int b = n + 1; b <= n_x;++b)
                if(st[b] == b && S[b] == 1)d = min(d, lab[b]/2);
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x]){
                    if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
                    else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
                }
            for(int u = 1; u <= n; ++u){
                if(S[st[u]] == 0){
                    if(lab[u] <= d)return 0;
                    lab[u] -= d;
                }else if(S[st[u]] == 1)lab[u] += d;
            }
            for(int b = n+1; b <= n_x; ++b)
                if(st[b] == b){
                    if(S[st[b]] == 0) lab[b] += d * 2;
                    else if(S[st[b]] == 1) lab[b] -= d * 2;
                }
            q=queue<int>();
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) ==
    ↪ 0)
                    if(on_found_edge(g[slack[x]][x]))return true;
            for(int b = n + 1; b <= n_x; ++b)
                if(st[b] == b && S[b] == 1 && lab[b] == 0)expand_blossom(b);
        }
        return false;
    }
    inline pair<long long, int> solve(){
        memset(match + 1, 0, sizeof(int) * n);
        n_x = n;
        int n_matches = 0;
        long long tot_weight = 0;
        for(int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();
        int w_max = 0;
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v){
                flower_from[u][v] = (u == v ? u : 0);
                w_max = max(w_max, g[u][v].w);
            }
        for(int u = 1; u <= n; ++u) lab[u] = w_max;
        while(matching()) ++n_matches;
        for(int u = 1; u <= n; ++u)
            if(match[u] && match[u] < u)
                tot_weight += g[u][match[u]].w;
        return make_pair(tot_weight, n_matches);
    }
    inline void init(){
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v)
                g[u][v]=edge(u, v, 0);
    }
};
```

必经点 dominator tree

```
//solve(s, n, raw_g): s is the root and base accords to base of raw_g
//idom[x] will be x if x does not have a dominator,and will be -1 if x is not reachable from
  ↪ s.

struct dominator_tree {
    int base, dfn[N], sdom[N], idom[N], id[N], f[N], fa[N], smin[N], stamp;
    Graph *g;
    void predfs(int u) {
        id[dfn[u] = stamp++] = u;
        for (int i = g -> adj[u]; ~i; i = g -> nxt[i]) {
            int v = g -> v[i];
            if (dfn[v] < 0) {
                f[v] = u;
                predfs(v);
            }
        }
    }
    int getfa(int u) {
        if (fa[u] == u) return u;
        int ret = getfa(fa[u]);
        if (dfn[sdom[smin[fa[u]]]] < dfn[sdom[smin[u]]])
            smin[u] = smin[fa[u]];
        return fa[u] = ret;
    }
    void solve (int s, int n, Graph *raw_graph) {
        g = raw_graph;
        base = g -> base;
        memset(dfn + base, -1, sizeof(*dfn) * n);
        memset(idom + base, -1, sizeof(*idom) * n);
        static Graph pred, tmp;
        pred.init(base, n);
        for (int i = 0; i < n; ++i) {
            for (int p = g -> adj[i + base]; ~p; p = g -> nxt[p])
                pred.ins(g -> v[p], i + base);
        }
        stamp = 0; tmp.init(base, n); predfs(s);
        for (int i = 0; i < stamp; ++i) {
            fa[id[i]] = smin[id[i]] = id[i];
        }
        for (int o = stamp - 1; o >= 0; --o) {
            int x = id[o];
            if (o) {
                sdom[x] = f[x];
                for (int i = pred.adj[x]; ~i; i = pred.nxt[i]) {
                    int p = pred.v[i];
                    if (dfn[p] < 0) continue;
                    if (dfn[p] > dfn[x]) {
                        getfa(p);
                        p = sdom[smin[p]];
                    }
                    if (dfn[sdom[x]] > dfn[p]) sdom[x] = p;
                }
                tmp.ins(sdom[x], x);
            }
            while (~tmp.adj[x]) {
                int y = tmp.v[tmp.adj[x]];
                tmp.adj[x] = tmp.nxt[tmp.adj[x]];
                getfa(y);
                if (x != sdom[smin[y]]) idom[y] = smin[y];
                else idom[y] = x;
            }
            for (int i = g -> adj[x]; ~i; i = g -> nxt[i])
                if (f[g -> v[i]] == x) fa[g -> v[i]] = x;
        }
        idom[s] = s;
        for (int i = 1; i < stamp; ++i) {
            int x = id[i];
            if (idom[x] != sdom[x]) idom[x] = idom[idom[x]];
        }
    }
};
```

Chapter 4

字符串

Manacher

```
//prime is the origin string(0-base)
//-10,-1,-20 are added to s
//length of s is exactly 2 * l + 3
inline void manacher(char prime[]) {
    int l = strlen(prime), n = 0;
    s[n++] = -10;
    s[n++] = -1;
    for (int i = 0; i < l; ++i) {
        s[n++] = prime[i];
        s[n++] = -1;
    }
    s[n++] = -20; f[0] = 1;
    int mx = 0, id = 0;
    for (int i = 1; i + 1 < n; ++i) {
        f[i] = i > mx ? 1 : min(f[id * 2 - i], mx - i + 1);
        while (s[i + f[i]] == s[i - f[i]]) ++f[i];
        if (i + f[i] - 1 > mx) {
            mx = i + f[i] - 1;
            id = i;
        }
    }
}
```

数组版后缀自动机

```
/*
 * Suffix Automaton - array version
 * SAMSAMSAM? SAMSAMSAM!
 */

```

```
namespace SAM {
    int to[100005 << 1][26], parent[100005 << 1], step[100005 << 1], tot;
    int root, np;
    int sam_len;

    int newnode(int STEP = 0) {
        ++tot;
        memset(to[tot], 0, sizeof to[tot]);
        parent[tot] = 0;
        step[tot] = STEP;
        return tot;
    }

    void init() {
        tot = 0;
        root = np = newnode(sam_len = 0);
    }

    void extend(char ch) {
        int x = ch - 'a';
        int last = np; np = newnode(++sam_len);
        for (; last && !to[last][x]; last = parent[last])
            to[last][x] = np;
        if (!last) parent[np] = root;
        else {
            int q = to[last][x];
            if (step[q] == step[last] + 1) parent[np] = q;
            else {
                nq = newnode(step[last] + 1);
                memcpy(to[nq], to[q], sizeof to[q]);
                parent[nq] = parent[q];
                parent[q] = parent[np] = nq;
                for (; last && to[last][x] == q; last = parent[last])
                    to[last][x] = nq;
            }
        }
    }
}
```

指针版后缀自动机

```
/*
 * Suffix Automaton - pointer version
 * SAMSAMSAM? SAMSAMSAM!
 */
```

```

namespace SAM {
    struct Node *pool_pointer;
    struct Node {
        Node *to[26], *parent;
        int step;

        Node(int STEP = 0): step(STEP) {
            memset(to, 0, sizeof to);
            parent = 0;
            step = 0;
        }

        void *operator new (size_t) {
            return pool_pointer++;
        }
    } pool[100005 << 1], *root, *np;
    int sam_len;

    void init() {
        pool_pointer = pool;
        root = np = new Node(sam_len = 0);
    }

    void extend(char ch) {
        static Node *last, *q, *nq;

        int x = ch - 'a';
        last = np; np = new Node(++sam_len);
        for (; last && !last->to[x]; last = last->parent)
            last->to[x] = np;
        if (!last) np->parent = root;
        else {
            q = last->to[x];
            if (q->step == last->step + 1) np->parent = q;
            else {
                nq = new Node(*q);
                nq->step = last->step + 1;
                q->parent = np->parent = nq;
                for (; last && last->to[x] == q; last = last->parent)
                    last->to[x] = nq;
            }
        }
    }
}
```

广义后缀自动机

```
/*
 * EX Suffix Automaton - pointer version
 * SAMSAMSAM? SAMSAMSAM!
 */

namespace SAM {
    struct Node *pool_pointer;
    struct Node {
        Node *parent, *to[26];
        int step;

        Node(int step = 0): step(step) {
            memset(to, 0, sizeof to);
            parent = 0;
        }

        void *operator new (size_t) {
            return pool_pointer++;
        }
    } pool[100005 * 10 << 1], *root, *np;
    int sam_len, now_len;

    void init() {
        sam_len = now_len = 0;
        pool_pointer = pool;
        root = new Node();
    }

    void new_str() { // a new string start
        now_len = 0;
        np = root;
    }

    void extend(char ch) {
        static Node *last, *q, *nq;

        int x = ch - 'a';
        if (np->to[x]) {
            np = np->to[x];
            ++now_len;
        }
        else {
            last = np; np = new Node(++now_len);
            for (; last && !last->to[x]; last = last->parent)
```

```
                last->to[x] = np;
            if (!last) np->parent = root;
            else {
                q = last->to[x];
                if (q->step == last->step + 1) np->parent = q;
                else {
                    nq = new Node(*q);
                    nq->step = last->step + 1;
                    q->parent = np->parent = nq;
                    for (; last && last->to[x] == q; last = last->parent)
                        last->to[x] = nq;
                }
            }
        }

        sam_len = std::max(sam_len, now_len);
    }
}
```

Chapter 5

技巧

无敌的读入优化

```
// getchar() 读入优化 << 关同步 cin << 此优化
// 用 isdigit() 会小幅变慢
// 返回 false 表示读到文件尾
namespace Reader {
    const int L = (1 << 15) + 5;
    char buffer[L], *S, *T;
    __inline bool getchar(char &ch) {
        if (S == T) {
            T = (S = buffer) + fread(buffer, 1, L, stdin);
            if (S == T) {
                ch = EOF;
                return false;
            }
        }
        ch = *S++;
        return true;
    }
    __inline bool getint(int &x) {
        char ch; bool neg = 0;
        for (; getchar(ch) && (ch < '0' || ch > '9'); ) neg ^= ch == '-';
        if (ch == EOF) return false;
        x = ch - '0';
        for (; getchar(ch), ch >= '0' && ch <= '9'; )
            x = x * 10 + ch - '0';
        if (neg) x = -x;
        return true;
    }
}
```

真正释放 STL 内存

```
template <typename T>
__inline void clear(T& container) {
    container.clear(); // 或者删除了一堆元素
    T(container).swap(container);
}
```

梅森旋转算法

```
template <typename T>
__inline void clear(T& container) {
    container.clear(); // 或者删除了一堆元素
    T(container).swap(container);
}
```

32-bit/64-bit 随机素数

32-bit	64-bit
73550053	1249292846855685773
148898719	1701750434419805569
189560747	3605499878424114901
459874703	5648316673387803781
1202316001	6125342570814357977
1431183547	6215155308775851301
1438011109	6294606778040623451
1538762023	6347330550446020547
1557944263	7429632924303725207
1981315913	8524720079480389849

NTT 素数及其原根

Prime	Primitive root
1053818881	7
1051721729	6
1045430273	3
1012924417	5
1007681537	3