代码模板存档

2022.9.30 增加并查集、埃氏筛、线性筛、快速幂、扩展欧几里得、exgcd 求逆元、费马小定理求逆元、线性递推求逆元

2022.10.11 增加树状数组

2022.10.12 增加用树状数组求逆序对

2022.10.21 增加超多东西，分类整理

2022.12.19 珂朵莉树

2023.1.25 字典树

2023.2.4 哈希

2023.3.2 网络流

2023.3.5 更新部分模板

一般 C++ 比赛文件模板

```cpp
#include <bits/stdc++.h>

using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);



    return 0;
}
```

数据结构

线段树

杨神（那里偷来的改过的）线段树模板

区间修改，区间查询

```cpp
template <class T>
struct SegmentTree {
    std::vector<T> val, add;
    int N;
#define ls (p * 2)
#define rs (p * 2 + 1)

    SegmentTree(int n = 0) {
        // 0 ~ N - 1
        N = 2 << std::__lg(n + 1);
        val.resize(N * 2);
        add.resize(N * 2);
    }
    void build(const std::vector<T> &a) {
        for (int i = 0; i < a.size(); i++)
            val[i + N] = a[i];
        for (int i = N - 1; i >= 1; i--)
            pull(i);
    }
    void modify(int l, int r, T x) {
        modify(l, r, x, 1, 0, N);
    }
    T query(int l, int r) {
        return query(l, r, 1, 0, N);
    }

private:
    void pull(int p) {
        val[p] = val[ls] + val[rs];
    }
    void push(int p, int len) {
        T &tag = add[p];
        if (tag) {
            add[ls] += tag;
            add[rs] += tag;
            val[ls] += tag * (len / 2);
            val[rs] += tag * (len / 2);
            tag = 0;
        }
    }
    void modify(int l, int r, T x, int p,
int L, int R) {
        if (l <= L && R <= r) {
            val[p] += x * (R - L);
            add[p] += x;
            return;
        }
        push(p, R - L);
        int M = (L + R) / 2;
        if (l < M)
            modify(l, r, x, ls, L, M);
        if (r > M)
            modify(l, r, x, rs, M, R);
        pull(p);
    }
    T query(int l, int r, int p, int L, in
t R) {
        if (l <= L && R <= r) {
            return val[p];
        }
        push(p, R - L);
        int M = (L + R) / 2;
        T v = T();
```

```
        if (l < M)
            v += query(l, r, ls, L, M);
        if (r > M)
            v += query(l, r, rs, M, R);
        return v;
    }
#undef ls
#undef rs
};
```

jiangly 那里偷来的模板

```
constexpr int N = 1 << 18;

int max[N], min[N];
i64 sum[N];

void pull(int p) {
    max[p] = std::max(max[2 * p], max[2 * p + 1]);
    min[p] = std::min(min[2 * p], min[2 * p + 1]);
    sum[p] = sum[2 * p] + sum[2 * p + 1];
}

void build(int p, int l, int r, auto &a) {
    if (r - l == 1) {
        // max[p] = min[p] = sum[p] = a[l];
        // if (a[l] == 0) max[p] = min[p] = 100;
        return;
    }
    int m = (l + r) / 2;
    build(2 * p, l, m, a);
    build(2 * p + 1, m, r, a);
    pull(p);
}

void modify(int p, int l, int r, int x, int y, int k) {
    // if (max[p] <= 100 && min[p] >= 99) return;
    if (l >= y || r <= x) return;
    if (r - l == 1) {
        // do sth
        return;
    }
    int m = (l + r) / 2;
    modify(2 * p, l, m, x, y, k);
    modify(2 * p + 1, m, r, x, y, k);
    pull(p);
}
```

并查集

```
struct DSU{
    std::vector<int> p, siz;
    DSU(int n) : p(n), siz(n, 1) { std::iota(p.begin(), p.end(), 0); }

    int leader(int x) {
        while (x != p[x]) {
            x = p[x] = p[p[x]];
        }
        return p[x];
    }

    void merge(int x, int y) {
        int px = leader(x);
        int py = leader(y);
        if (px == py) {
            return ;
        }
        if (siz[px] <= siz[py]) {
            siz[py] += siz[px];
            p[px] = py;
        } else {
            siz[px] += siz[py];
            p[py] = px;
        }
    }

    bool same(int x, int y) { return leader(x) == leader(y); }

    int size(int x) { return siz[leader(x)]; }
};
```

树状数组

单点修改，区间求和

树状差分数组：区间修改，单点查询

```
template<typename T>
struct FenwickTree {
    int n;
    std::vector<T> bit;
    FenwickTree(int n) : n(n), bit(n + 1) {}

    void add(int idx, T val) {
        for (; idx <= n; idx |= idx + 1) {
            bit[idx] += val;
        }
    }

    T sum(int r) {
```

```
        T res = 0;
        for (; r > 0; r = (r & (r + 1)) - 1) {
            res += bit[r];
        }
        return res;
    }

    T query(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};
```

- T 的可用类型： int, long long, unsigned int, unsigned long long, modInt
- $0 < n \leq 10^8$
- query(int L, int R) 用于求 $\sum_{i=L}^{R} a_i$

树状数组求逆序对

（接上）

```
int getInv(std::vector<int>&a) {
    int n = a.size();
    Fenwick fen(n);
    int ans = 0;
    for (int i = n - 1; i >= 0; i--) {
        ans += fen.sum(a[i]);
        fen.add(a[i], 1);
    }
    return ans;
}
```

珂朵莉树

jiangly std::map 版本

```
struct ODT {
    const int n;
    std::map<int, int> mp;
    ODT(int n) : n(n) { mp[-1] = 0; }

    void split(int x) {
        // 以 x 为左端点
        auto it = prev(mp.upper_bound(x));

        mp[x] = it->second;
    }
    void assign(int l, int r, int v) {
        // 删除 [l, r - 1] 之间的左端点
        // r 是右端点+1
        split(l);
        split(r);
        auto it = mp.find(l);
        while (it->first != r) {
            it = mp.erase(it);
        }
        mp[l] = v;
    }
    int sum(int l, int r) {
        int res = 0;
        auto it = mp.find(l);
        while (it->first != r) {
            res += it->second * (std::next(it)->first - it->first);
            it = std::next(it);
        }
        return res;
    }
    void update(int l, int r, int c) {
        split(l);
        split(r);
        auto it = mp.find(l);
        while (it->first != r) {
            //
            it = next(it);
        }
    }
};
```

字典树 Trie

```
const int N = 1E5 + 10;

int cnt = 1;
int trie[N][26], c[N];

int newNode() {
    int x = ++cnt;
    // initiallization
    return x;
}

void add(const std::string &s) {
    int p = 1;
    for (auto &ch : s) {
        int u = ch - 'a';
        if (!trie[p][u]) trie[p][u] = newNode();
        p = trie[p][u];
    }

    c[p]++;
}

int query(const std::string &s) {
    int p = 1;
    for (auto &ch : s) {
```

```
        int u = ch - 'a';
        if (!trie[p][u]) return 0;
        p = trie[p][u];
    }
    return c[p];
}
```

01trie

```
const int N = 1E5 + 10;

int cnt = 1;
int trie[31 * N][2];

int newNode() {
    int x = ++cnt;
    trie[x][0] = trie[x][1] = 0;
    return x;
}

void add(int x) {
    int p = 1;
    for (int i = 30; i >= 0; i--) {
        int d = (x >> i) & 1;
        if (!trie[p][d]) trie[p][d] = newN
ode();
        p = trie[p][d];
    }
}

int query(int x) {
    int p = 1;
    // int ans = 0;
    for (int i = 30; i >= 0; i--) {
        int d = (x >> i) & 1;
        if (trie[p][!d]) {
            // do sth.
            p = trie[p][!d];
        } else {
            p = trie[p][d];
        }
    }
    return ans;
}
```

使用时先初始化：

```
cnt = 0;
newNode();
```

数学相关

埃氏筛

```
std::vector<int> Eratosthenes(int mx) {
    std::vector<int> plist;
    std::vector<bool> fl(mx + 1, false);
    for(int i = 2; i <= mx; i++) {
        if(fl[i]) {
            continue;
        }
        plist.push_back(i);
        for(int j = i; j <= mx; j += i) {
            fl[j] = true;
        }
    }

    return plist;
}
```

线性筛

```
std::vector<int> conplist(int n) {
    std::vector<bool> f(n + 1, 0);
    std::vector<int> plist;
    for (int i = 2; i <= n; i++) {
        if (!f[i]) {
            plist.push_back(i);
        }
        for (int j = 0; plist[j] <= n / i;
 j++) {
            f[plist[j] * i] = 1;
            if (i % plist[j] == 0) {
                break;
            }
        }
    }

    return plist;
}
```

快速幂

```
int power(i64 a, i64 b, int mod) {
    i64 res = 1;
    for (; b; a *= a, a %= mod, b *= 2, b %
= mod) {
        if (b & 1) {
            res *= a;
            res %= mod;
        }
    }
    return res;
}
```

扩展欧几里得

求形如 $a \equiv 1\ (mod\ m)$

即对于不定方程 $ua + vb \equiv 1(mod\ m)$, 求出绝对值最小的 u

```
int exgcd(int a, int m, int& u, int& v) {
    if(m == 0) {
        u = 1; v = 0;
        return a;
    } else {
        int x1;
        int d = exgcd(m, a % m, x1, u);
        v = x1 - a / m * u;
        return d;
    }
}
```

exgcd 求乘法逆元

求 a 关于 p 的逆元，即求 $a \equiv 1\ (mod\ p)$ 下，a 的逆元

```
int inv(int a, int p) {
    int u, v;
    exgcd(a, p, u, v);
    return (u % p + p) % p;
}
```

费马小定理求逆元

当 p 是质数时，有：

$$a^{p-1} \equiv 1\ (mod\ p)$$

也即

$$a^{p-2} \equiv a^{-1}\ (mod\ p)$$

```
int fermatInv(int a, int b, int mod) {
    a %= mod;
    int ans = 1;
    while (b != 0) {
        if (b % 2 == 1) {
            ans *= a;
        }
        a *= a;
        b >>= 1;
    }
    return ans;
}
```

使用时 `res = fermatInv(a, p - 2, p)`

线性推求逆元

求 1 到 n 中所有数模一个数字 p 的逆元

```
std::vector<int> linear_inv(int n, int mod)
 {
    std::vector<int> inv(n + 1);
    inv[1] = 1;
    for (int i = 2; i <= n; i++) {
        inv[i] = (mod - mod / i) * inv[mod
 % i] % mod;
    }
    return inv;
}
```

大数取模/运算模板

从 jls 那里偷来的板子

```
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    Z(i64 x) : x(norm(x % P)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
```

```
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const
Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const
Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const
Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const
Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
    friend std::istream &operator>>(std::i
stream &is, Z &a) {
        i64 v;
        is >> v;
        a = Z(v);
        return is;
    }
    friend std::ostream &operator<<(std::o
stream &os, const Z &a) {
        return os << a.val();
    }
};
```

BSGS

求解形如 $a^x \equiv b \ (mod\ m)$

其中 a, b 互质

```
int BSGS(int a, int b, int m) {
    std::unordered_map<int, int> mp;
    int cur = 1, t = std::sqrt(m) + 1;
    for (int B = 1; B <= t; B++) {
        cur = (cur * a) % m;
        mp[(b * cur) % m] = B;
    }

    int curr = cur;
    for (int A = 1; A <= t; A++) {
        auto it = mp.find(curr);
        if (it != mp.end()) {
            return A * t - it->second;
        }
        curr = (curr * cur) % m;
    }

    return -1;
}
```

图论

Kruskal

```
struct Edge {
    int u, v, d;

    bool operator < (const edge &t) const
{
        return val < t.val;
    }
};


// 并查集
// 记 std::vector<Edge> E;
// 共 n 个点;

std::sort(E.begin(), E.end());

DSU dsu(n);
int ans = 0; // 总长度
int cnt = 0; // 总边数
for (auto[u, v, d] : E) {
    if (!dsu.same(u, v)) {
        dsu.merge(u, v);
        ans += d;
        cnt++;
    }
}

if (cnt < n - 1) {
    // 不存在连通
```

```
}
```

如果不重载运算符就用

```
std::sort(E.begin(), E.end(), [](Edge x, E
dge y) {
    return x.d > y.d;
});
```

Prim

适合稠密图求最小生成树

```
std::vector<std::vector<int>> g(n, std::ve
ctor<int>(n, 0x3f3f3f3f));
for (int i = 0; i < m; i++) {
    int a, b, d;
    std::cin >> a >> b >> d;
    a--, b--;
    g[a][b] = g[b][a] = std::min(g[a][b],
d);
}

// prim
std::vector<bool> vis(n, 0);
std::vector<int> dist(n, 0x3f3f3f3f);

int res = 0; // 边权和

for (int i = 0; i < n; i++) {
    int t = -1; // 当前的点
    for (int j = 0; j < n; j++) {
        if (!vis[j] && (t == -1 || dist[j]
 < dist[t])) {
            t = j;
        }
    }
    if (i && dist[t] == 0x3f3f3f3f) {
        std::cout << "impossible\n";
        return 0;
    }
    if (i) {
        res += dist[t];
    }

    for (int j = 0; j < n; j++) {
        dist[j] = std::min(dist[j], g[t]
[j]);
    }

    vis[t] = 1;
}
```

bellman-ford

存在负权边的情况下

用 vector 存图：

```
std::vector<std::array<int, 3>> edges(m);
for (int i = 0; i < m; i++) {
    int a, b, d;
    std::cin >> a >> b >> d;
    edges[i] = {a - 1, b - 1, d};
}

void bellman_ford() {
    std::vector<int> dist(n, 0x3f3f3f3f);
    dist[0] = 0;

    for (int i = 0; i < k; i++) {
        std::vector<int> backup(dist.begin
(), dist.end());
        for (int j = 0; j < m; j++) {
            int a = edges[j][0], b = edges
[j][1], d = edges[j][2];
            dist[b] = std::min(dist[b], ba
ckup[a] + d);
        }
    }

    if (dist[n - 1] > 0x3f3f3f3f / 2) {
        // 不存在
    }
}
```

Floyd 最短路

用邻接矩阵存图

```
std::vector<std::vector<int>> g(n, std::ve
ctor<int>(n, 1E9));
for (int i = 0; i < n; i++) {
    g[i][i] = 0;
}

for (int i = 0; i < m; i++) {
    int a, b, d;
    std::cin >> a >> b >> d;
    a--, b--;
    g[a][b] = std::min(g[a][b], d);
}
```

核心部分

```
for (int l = 0; l < n; l++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            g[i][j] = std::min(g[i][j], g
```

```
[i][l] + g[l][j]);
            }
        }
    }
```

染色法判二分图

用 vector 存图

```
std::vector<std::vector<int>> g(n);
for (int i = 0; i < m; i++) {
    int u, v;
    std::cin >> u >> v;
    u--, v--;
    g[u].push_back(v);
    g[v].push_back(u);
}
```

核心部分

```
std::vector<int> st(n, 0);

std::function<bool(int, int)> dfs = [&](in
t u, int color) -> bool {
    st[u] = color;
    for (auto p : g[u]) {
        if (!st[p] && !dfs(p, 3 - color))
{
            // if not colored
            return false;
        } else if (st[p] == color) {
            return false;
        }
    }
    return true;
};

for (int i = 0; i < n; i++) {
    if (!st[i] && !dfs(i, 1)) {
        // 不是二分图
        return 0;
    }
}

//能走到这里说明 是二分图
```

二分图的最大匹配-匈牙利算法

vector 存图

```
// 左 n1 个点，右 n2 个点
// m 条边
int n1, n2, m;
std::cin >> n1 >> n2 >> m;
std::vector<std::vector<int>> g(n1);
for (int i = 0; i < m; i++) {
    int u, v;
    std::cin >> u >> v;
    u--, v--;
    g[u].push_back(v);
}
```

核心部分

```
std::vector<int> match(n2, -1);
std::vector<bool> vis(n2);

std::function<bool(int)> find = [&](int x)
 -> bool {
    for (auto j : g[x]) {
        if (!vis[j]) {
            vis[j] = true;
            if (match[j] == -1 || find(mat
ch[j])) {
                match[j] = x;
                return true;
            }
        }
    }
    return false;
};

int ans = 0;
for (int i = 0; i < n1; i++) {
    std::fill(vis.begin(), vis.end(), 0);
    if (find(i)) {
        ans++;
    }
}
// 最大匹配即为 ans
```

网络流

```
struct Flow {
    static constexpr int INF = 1E9;
    int n;
    struct Edge {
        int to, cap;
        Edge(int to, int cap) : to(to), ca
p(cap) {}
    };
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;
    Flow(int n) : n(n), g(n) {}
    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> que;
        h[s] = 0;
        que.push(s);
```

```
        while (!que.empty()) {
            int u = que.front();
            que.pop();
            for (int i : g[u]) {
                int v = e[i].to;
                int c = e[i].cap;
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) return tru
e;
                    que.push(v);
                }
            }
        }
        return false;
    }
    int dfs(int u, int t, int f) {
        if (u == t) return f;
        int r = f;
        for (int &i = cur[u]; i < int(g[u].
size()); ++i) {
            int j = g[u][i];
            int v = e[j].to;
            int c = e[j].cap;
            if (c > 0 && h[v] == h[u] + 1)
 {
                int a = dfs(v, t, std::min
(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) return f;
            }
        }
        return f - r;
    }
    void addEdge(int u, int v, int c) {
        g[u].push_back(e.size());
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    int maxFlow(int s, int t) {
        int ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, INF);
        }
        return ans;
    }
};
//Flow flow(n + 3);
```

计算几何（待施工）

jiangly 那边偷来的板子

```
struct Point {
    double x;
    double y;
    Point(double x = 0, double y = 0) : x
(x), y(y) {}

    Point &operator+=(const Point &p) {
        x += p.x, y += p.y;
        return *this;
    }
    Point &operator-=(const Point &p) {
        x -= p.x, y -= p.y;
        return *this;
    }
    Point &operator*=(const double &v) {
        x *= v, y *= v;
        return *this;
    }
    friend Point operator-(const Point &p)
 {
        return Point(-p.x, -p.y);
    }
    friend Point operator+(Point lhs, cons
t Point &rhs) {
        return lhs += rhs;
    }
    friend Point operator-(Point lhs, cons
t Point &rhs) {
        return lhs -= rhs;
    }
    friend Point operator*(Point lhs, cons
t double &rhs) {
        return lhs *= rhs;
    }
};

double Dot(const Point &a, const Point &b)
 {
    return a.x * b.x + a.y * b.y;
}

double Cross(const Point &a, const Point &
b) {
    return a.x * b.y - a.y * b.x;
}
```

字符串

KMP

核心代码

```
void get_pmt(const string& s) {
    for (int i = 1, j = 0; i < s.length();
 ++i) {
        while (j && s[i] != s[j]) j = pmt
[j - 1];
        if (s[i] == s[j]) j++;
        pmt[i] = j;
    }
}
void kmp(const string& s, const string& p)
 {
    for (int i = 0, j = 0; i < s.length();
 ++i) {
        while (j && s[i] != p[j]) j = pmt
[j - 1];
        if (s[i] == p[j]) j++;
        if (j == p.length()) {
            j = pmt[j - 1];
        }
    }
}
```

使用前

```
std::string s, p;
std::cin >> s >> p;
get_pi(p);
kmp(s, p);
```

字典树 Trie

见“数据结构”

哈希

常用哈希常数

```
const i64 BASE1 = 2333, BASE2 = 13331;
const i64 MOD1 = 998244353, MOD2 = 1E9 + 7;
```

或

```
const i64 BASE[] = {2333, 13331};
const i64 MOD[] = {998244353, 1000000007};
```