default.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define lli long long
//#define int long long
inline void _QuickStreamOpen(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
}
const bool _QuickStream=true;
const bool _FILE=false;
const int  _TEST=0;
//-----------------------------------------

signed main(){
    if(_QuickStream){_QuickStreamOpen();}
    if(_FILE){
        freopen(".in","r",stdin);
        freopen(".out","w",stdout);
    }

    return 0;
}
```

链式前向星

```cpp
// head[u] 和 cnt 的初始值都为 -1
void add(int u, int v) {
  nxt[++cnt] = head[u];  // 当前边的后继
  head[u] = cnt;         // 起点 u 的第一条边
  to[cnt] = v;           // 当前边的终点
}

// 遍历 u 的出边
for (int i = head[u]; ~i; i = nxt[i]) {  // ~i 表示 i != -1
  int v = to[i];
}
```

拓扑排序

```cpp
const int maxn = 110000;
int n, m, degree[maxn];
vector<int> G[maxn];
```

```
vector<int> toposort1() {
    queue<int> Q;
    for (int i = 1; i <= n; ++i)
    degree[i] = 0;
    for (int x = 1; x <= n; ++x)
    for (auto y : G[x])
    degree[y]++;
    for (int i = 1; i <= n; ++i) if (degree[i] == 0)
    Q.push(i);
    vector<int> res;
    while (!Q.empty()) {
        int x = Q.front(); Q.pop();
        res.push_back(x);
        for (auto y : G[x]) {
            degree[y]--;
            if (degree[y] == 0)
            Q.push(y);
        }
    }
    return res;
}

vector<int> result;
int vis[maxn];

void dfs(int x) {
    vis[x] = true;
    for (auto y : G[x]) if (!vis[y])
    dfs(y);
    result.push_back(x);
}

vector<int> toposort2() { //必须保证是有向无环图
    memset(vis, 0, sizeof(vis));
    for (int i = 1; i <= n; ++i) if (!vis[i])
    dfs(i);
    reverse(result.begin(), result.end());
    return result;
}

int main() {//uva 10305
    while (scanf("%d %d", &n, &m) == 2) {
        if (n == 0 && m == 0)
        break;
        for (int i = 1; i <= n; ++i)
        G[i].clear();
        for (int i = 0; i < m; ++i) {
            int x, y;
            scanf("%d %d", &x, &y);
            G[x].push_back(y);
        }
        auto ans = toposort2();
        printf("%d", ans[0]);
        for (int i = 1; i < n; ++i)
```

```
            printf(" %d", ans[i]);
            printf("\n");
        }
        return 0;
    }
```

Dijkstra 算法

```
int n, m, s; //点数、边数、起点
const int maxn = 210000;
vector<pair<int, int>> G[maxn];
int d[maxn];

void dijkstra() {
    using node = pair<int, int>;
    priority_queue<node, vector<node>, greater<node>> Q;
    memset(d, 0x3f, sizeof(d));
    d[s] = 0;
    Q.emplace(0, s);
    while (!Q.empty()) {
        auto [dist, x] = Q.top(); Q.pop();
        if (dist != d[x])
        continue;
        for (auto [y, w] : G[x]) {
            if (d[y] > d[x] + w) {
                d[y] = d[x] + w;
                Q.emplace(d[y], y);
                //p[y] = x;
            }
        }
    }
}
void solve() {
    scanf("%d %d %d", &n, &m, &s);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        G[x].emplace_back(y, z)
    }
}
```

倍增求 LCA

```
const int maxn = 510000;
const int maxlog = 20;
vector<int> G[maxn];
int anc[maxn][maxlog], dep[maxn];
    void dfs(int x, int fa, int d) {
```

```
    anc[x][0] = fa;
    dep[x] = d;
    for (auto y : G[x]) if (y != fa)
    dfs(y, x, d + 1);
}
void preprocess(int n) { //点的编号从 1 开始
    for (int j = 1; j < maxlog; ++j)
    for (int i = 0; i <= n; ++i)
    anc[i][j] = 0;
    dfs(1, 0, 0);
    for (int j = 1; j < maxlog; ++j)
    for (int i = 1; i <= n; ++i)
    anc[i][j] = anc[anc[i][j- 1]][j- 1];
}
//返回结点 x 向上走 d 步到达的结点
int moveup(int x, int d) {
    for (int i = 0; d >> i; ++i)
    if (d >> i & 1)
    x = anc[x][i];
    return x;
}
int lca(int x, int y) {
    if (dep[x] < dep[y])
    swap(x, y);
    x = moveup(x, dep[x]- dep[y]);
    if (x == y)
    return x;
    for (int i = maxlog- 1; i >= 0;--i)
    if (anc[x][i] != anc[y][i])
    x = anc[x][i], y = anc[y][i];
    return anc[x][0];
}
int dist(int x, int y) { //返回结点 x 和 y 之间的距离
    return dep[x] + dep[y]- 2 * dep[lca(x, y)];
}
int move(int x, int y, int d) { //返回从结点 x 向结点 y 走 d 步到达的结点
    int p = lca(x, y);
    int h = dep[x]- dep[p];
    if (h >= d)
    return moveup(x, d);
    else
    return moveup(y, dep[x] + dep[y]- d- 2 * dep[p]);
}
int main() {
    //freopen("in.txt", "r", stdin);
    int n, m, s;
    scanf("%d %d %d", &n, &m, &s);
    for (int i = 1; i < n; ++i) {
        int x, y;
        scanf("%d %d", &x, &y);
        G[x].push_back(y);
        G[y].push_back(x);
    }
    dfs(s, 0, 0);
    2 树算法
    35
    for (int j = 1; j < maxlog; ++j)
    for (int i = 1; i <= n; ++i)
    anc[i][j] = anc[anc[i][j- 1]][j- 1];
    for (int i = 0; i < m; ++i) {
        int x, y;
        scanf("%d %d", &x, &y);
        printf("%d\n", lca(x, y));
    }
    return 0;
}
```

哈希表

```
//key_t 应当为整数类型，且实际值必须非负
template<typename key_t, typename type> struct hash_table {
    static const int maxn = 1000010;
    static const int table_size = 11110007;
    int first[table_size], nxt[maxn], sz; //init: memset(first, 0, sizeof(first)),
sz = 0
    key_t id[maxn];
    type data[maxn];
    type& operator[] (key_t key) {
        const int h = key % table_size;
        for (int i = first[h]; i; i = nxt[i])
        if (id[i] == key)
        return data[i];
        int pos = ++sz;
        nxt[pos] = first[h];
        first[h] = pos;
        id[pos] = key;
        return data[pos] = type();
    }
    bool count(key_t key) {
        for (int i = first[key % table_size]; i; i = nxt[i])
        if (id[i] == key)
        return true;
        return false;
    }
    type get(key_t key) { //如果 key 对应的值不存在，则返回 type()。
        for (int i = first[key % table_size]; i; i = nxt[i])
        if (id[i] == key)
        return data[i];
        return type();
    }
};
unordered_map<long long, long long> A;
hash_table<long long, long long> B;
const int maxn = 1000000;
int main() {
    default_random_engine e;
    uniform_int_distribution<long long> d(0, LLONG_MAX);
    for (int i = 0; i < maxn; ++i) {
        long long key = d(e);
        long long value = d(e);
        A[key] = value;
        B[key] = value;
    }
    for (int i = 0; i < maxn * 10; ++i) {
        long long key = d(e);
        if (A.count(key) != B.count(key))
        abort();
        if (A.count(key)) {
            if (A[key] != B.get(key))
            abort();
        }
    }
    return 0;
}
```

计时器

```
struct Timer {
    std::chrono::steady_clock::time_point start;
    Timer() : start(std::chrono::steady_clock::now()) {}
    ~Timer() {
        auto finish = std::chrono::steady_clock::now();
        auto runtime = std::chrono::duration_cast<std::chrono::microseconds>
(finishstart).count();
        std::cerr << runtime / 1e6 << "s" << std::endl;
    }
};

int main() {
Timer timer;
std::set<int> S;
for (int i = 0; i < 1e6; ++i) {
S.insert(i);
}
return 0;
}
```

高精(其他板子)

```
typedef long long ll;
const int base = 100000000;
const int num_digit = 8;
const int maxn = 1000;
```

```
ll mul_mod (ll x, ll y, ll n){
    ll T = floor(sqrt(n) + 0.5);
    ll t = T * T- n;
    ll a = x / T; ll b = x % T;
    ll c = y / T; ll d = y % T;
    ll e = a * c / T; ll f = a * c % T;
    ll v = ((a * d + b * c) % n + e * t) % n;
    ll g = v / T; ll h = v % T;
    ll ans = (((f + g) * t % n + b * d) % n + h * T) % n;
    while (ans < 0) ans += n;
    return ans;
}
struct bign {
    int len;
    int s[maxn];
    bign(const char *str = "0"){ (*this) = str; }
    bign operator= (const char *str){
        int i;
        int j = strlen(str)- 1;
        len = j / num_digit + 1;
        for(i = 0; i <= len ; i++) s[i] = 0;
        for(i = 0; i <= j; i++){
            int k = (j- i) / num_digit + 1;
            s[k] = s[k] * 10 + str[i]- '0';
        }
        return *this ;
    }
};
void print(const bign &a){
    printf("%d", a.s[a.len]);
    for(int i = a.len- 1; i >= 1; i--)
    printf("%0*d", num_digit, a.s[i]);
}
//比较的前提是整数没有前导 0
int compare (const bign &a, const bign &b){
    if(a.len > b.len) return 1;
    if(a.len < b.len) return-1;
    int i = a.len;
    while ((i > 1) && (a.s[i] == b.s[i])) i--;
    return a.s[i]- b.s[i];
}
inline bool operator< (const bign &a, const bign &b) {
    return compare(a, b) < 0;
}
inline bool operator<= (const bign &a, const bign &b) {
    return compare(a, b) <= 0;
}
inline bool operator== (const bign &a, const bign &b) {
    return compare(a, b) == 0;
}
//加法和减法很容易写出，只需注意不要忽略前导 0
bign operator+ (const bign &a, const bign &b){
    bign c;
    int i;
    for(i = 1; i <= a.len || i <= b.len || c.s[i]; i++){
        if(i <= a.len) c.s[i] += a.s[i];
        if(i <= b.len) c.s[i] += b.s[i];
        c.s[i+1] = c.s[i] / base;
        c.s[i] %= base;
    }
    c.len = i-1;
    if(c.len == 0) c.len = 1;
    return c;
}
//减法的前提是 a > b
bign operator- (const bign &a, const bign &b){
    bign c;
    int i, j;
    for(i = 1, j = 0; i <= a.len; i++){
        c.s[i] = a.s[i]- j;
        if(i <= b.len ) c.s[i]-= b.s[i];
        if(c.s[i] < 0){ j = 1; c.s[i] += base; }
        else j = 0;
    }
    c.len = a.len ;
    while (c.len > 1 && ! c.s[c.len]) c.len--;
    return c;
}
bign operator* (const bign &a, const bign &b){
    bign c;
    ll g = 0;
    int i, k;
    c.len = a.len + b.len;
    c.s[0] = 0;
    for(i = 1; i <= c.len; i++) c.s[i] = 0;
    for(k = 1; k <= c.len; k++){
        ll tmp = g;
        i = k + 1- b.len;
        if(i < 1) i = 1;
        for (; i <= k && i <= a.len; i++)
        tmp += (ll)a.s[i] * (ll)b.s[k+1-i];
        g = tmp / base;
        c.s[k] = tmp % base;
    }
    while (c.len > 1 && !c.s[c.len]) c.len--;
    return c;
}
bign operator/ (const bign &a, int n) {
    ll g = 0;
    bign c;
    c.len = a.len;
    for (int i = a.len; i > 0;--i) {
        ll tmp = g * base + a.s[i];
        c.s[i] = tmp / n;
        g = tmp % n;
    }
    while (c.len > 1 && !c.s[c.len]) c.len--;
    return c;
```

```
}
bign operator/ (const bign &a, const bign &b) {
    bign L = "0", R = a;
    while (L < R) {
        bign M = L + (R- L + "1") / 2;
        if (M * b <= a) L = M;
        else R = M- "1";
    }
    return L;
}
ll bigmod(const bign &a, ll m){
    ll d = 0;
    for(int i = a.len; i > 0;--i){
        d = mul_mod(d, base, m);
        d = (d + a.s[i]) % m;
    }
    return d;
}
bign sqrt(const bign &n){
    bign c, d, x, y = n;
    do
    {
        x = y;
        y = (x + n / x) / 2;
    }
    while (y < x);
    return x;
}
bign gcd(bign a, bign b) {
    bign c = "1";
    for (;;) {
        if (a == b)
        return a * c;
        else if (a.s[1] % 2 == 0 && b.s[1] % 2 == 0) {
            a = a / 2;
            b = b / 2;
            c = c * "2";
        }
        else if (a.s[1] % 2 == 0) {a = a / 2;}
        else if (b.s[1] % 2 == 0) {b = b / 2;}
        else if (b < a) {a = a- b;}
        else{b = b- a;}
    }
}
int main() {
    bign a("345345345436546"), b("26768"), c, d;
    //divide(a, b, c, d);
    c = gcd(a, b);
    print(c);
    //print(a*b);
    //cout << '\n' << 3445453953435LL * 897676LL;
    return 0;
}
```

树状数组

```
#define lowbit(x) (x&-x)
namespace dimension1 {
constexpr int maxn = 550000;
int n; // 树状数组的长度
long long C[maxn];
inline auto sum(int x) { //计算前缀和
long long ret = 0;
for (int i = x; i > 0; i-= lowbit(i)) {
ret += C[i];
}
return ret;
}
inline void add(int x, int d) { //单点修改
for (int i = x; i <= n; i += lowbit(i)) {
C[i] += d;
}
}
}
namespace dimension2 {
const int maxn = 1100;
int n, m; // 树状数组的长度
long long C[maxn][maxn];
inline auto sum(int x, int y) { //计算二维前缀和
long long ret = 0;
for (int i = x; i > 0; i-= lowbit(i)) {
for (int j = y; j > 0; j-= lowbit(j)) {
ret += C[i][j];
}
}
return ret;
}
inline void add(int x, int y, int d) { //单点修改
for (int i = x; i <= n; i += lowbit(i)) {
for (int j = y; j <= m; j += lowbit(j)) {
C[i][j] += d;
}
}
}
}
int main() { // 洛谷 P3374
using namespace dimension1;
int m;
scanf("%d %d", &n, &m);
for (int i = 1; i <= n; ++i) {
int v;
scanf("%d", &v);
add(i, v);
}
while (m--) {
```

```
int tp, x, y;
scanf("%d %d %d", &tp, &x, &y);
if (tp == 1) {
add(x, y);
}
else {
printf("%lld\n", sum(y)- sum(x- 1));
}
}
return 0;
}
```

李超线段树

```
const int maxn = 200005;
const int inf = 1 << 30;
int cur = 0;
struct segment {
    int l, r, lc, rc;
    double k, b;
}t[maxn * 4];
void init() {cur = 0;}
int build(int L, int R) { //新建一棵线段树，并返回根结点的编号
    int p = ++cur;
    t[p].l = L;
    t[p].r = R;
    if (L < R) {
        int mid = (L + R) >> 1;
        t[p].lc = build(L, mid);
        t[p].rc = build(mid + 1, R);
    }
    t[p].k = 0; t[p].b =-inf;
    return p;
}
void pushdown(int p, double k, double b) {
    double l1 = k * t[p].l + b, r1 = k * t[p].r + b;
    double l2 = t[p].k * t[p].l + t[p].b, r2 = t[p].k * t[p].r + t[p].b;
    if (l1 >= l2 && r1 >= r2)
    t[p].k = k, t[p].b = b;
    else if (l2 < l1 || r2 < r1) {
        double pos = (b- t[p].b) / (t[p].k- k);
        int mid = (t[p].l + t[p].r) >> 1;
        if (pos <= mid) {
            if (r1 > r2)
            swap(t[p].k, k), swap(t[p].b, b);
            pushdown(t[p].lc, k, b);
        }
        else {
            if (l1 > l2)
            swap(t[p].k, k), swap(t[p].b, b);
            pushdown(t[p].rc, k, b);
        }
    }
}
void insert(int p, int L, int R, double k, double b) { //在区间 [L, R] 中插入直线 y
= kx + b
    if (L <= t[p].l && R >= t[p].r)
    pushdown(p, k, b);
    else {
        int mid = (t[p].l + t[p].r) >> 1;
        if (L <= mid)
        insert(t[p].lc, L, R, k, b);
        if (R > mid)
        insert(t[p].rc, L, R, k, b);
    }
}
double query(int p, int x) { //p 是线段树的根结点, x 是查询的横坐标, 返回所有直线在 x
处的最大值
    double ans = t[p].k * x + t[p].b;
    if (t[p].l < t[p].r) {
        int mid = (t[p].l + t[p].r) >> 1;
        if (x <= mid)
        ans = max(ans, query(t[p].lc, x));
        else
        ans = max(ans, query(t[p].rc, x));
    }
    return ans;
}
const double eps = 1e-9;
struct Seg {
    Seg() : k(), b(), id(1) {}
    Seg(double k, double b) : k(k), b(b) {}
    double k, b;
    int id;
}A[maxn];
    int main() {
    srand(time(0));
    int n = 200000, m = 300;
    for (int i = 1; i <= m; ++i)
    A[i].k = rand() % 10 + 1, A[i].b = rand() % 1000- 500, A[i].id = i;
    init();
    int root = build(1, n);
    for (int i = 1; i <= m; ++i)
    insert(root, 1, n, A[i].k, A[i].b);
    for (int i = 1; i <= n; ++i) {
        long long ans = query(root, i) + eps;
        long long res =-inf;
        for (int j = 1; j <= m; ++j)
        res = max(res, (long long)(A[j].k * i + A[j].b + eps));
        if (res != ans)
        printf("%lld %lld\n", ans, res);
    }
    return 0;
}
```

线段树

```
const int maxn = 210000, offset = 210000;
const int inf = 1 << 30;
struct tmp {
    int data[maxn * 5];
    int& operator[] (int idx) {
        return data[idx + offset];
    }
}A;
#define lc t[p].lchild
#define rc t[p].rchild
int cur = 0, tot, mn, mx;
struct segment {
    int l, r, lchild, rchild;
    int sum, min, max, set, add;
}t[maxn * 4];
void init() {
    cur = 0;
}
inline void maintain(int p) {
    t[p].sum = t[lc].sum + t[rc].sum;
    t[p].min = min(t[lc].min, t[rc].min);
    t[p].max = max(t[lc].max, t[rc].max);
}
inline void mark(int p, int setv, int addv) { //给结点打标记
    if (setv >= 0) {
        t[p].set = setv; t[p].add = 0;
        t[p].min = t[p].max = setv;
        t[p].sum = setv * (t[p].r- t[p].l + 1);
    }
    if (addv) {
        t[p].add += addv;
        t[p].min += addv;
        t[p].max += addv;
        t[p].sum += addv * (t[p].r- t[p].l + 1);
    }
}
inline void pushdown(int p) { //pushdown 将标记传递给子结点，不影响当前结点的信息。
    mark(lc, t[p].set, t[p].add);
    mark(rc, t[p].set, t[p].add);
    t[p].set =-1;
    t[p].add = 0;
}
→
int build(int L, int R) { //只要计算 mid 的方式是 (L + R) >> 1 而不是 (L + R) / 2,
就可以建立负坐标线段树。
    int p = ++cur;
    t[p].l = L;
    t[p].r = R;
    t[p].add = 0; t[p].set =-1; //清空结点标记
    if (t[p].l == t[p].r) {
        mark(p, 0, A[L]);
    }
    else {
        int mid = (t[p].l + t[p].r) >> 1;
        lc = build(L, mid);
        rc = build(mid + 1, R);
        maintain(p);
    }
    return p;
}
void update(int p, int L, int R, int op, int v) {
    if (L <= t[p].l && R >= t[p].r) {
        if (op == 0)
        mark(p,-1, v);
        else
        mark(p, v, 0);
    }
    else {
        pushdown(p); //如果没有 pushdown 只需要在最后调用一次 maintain 即可。
        int mid = (t[p].l + t[p].r) >> 1;
        if (L <= mid)
        update(lc, L, R, op, v);
        if (R > mid)
        update(rc, L, R, op, v);
        maintain(p);
    }
}
void update(int p, int pos, int v) { //单点修改
    if (t[p].l == t[p].r) {
        mark(p,-1, v);
    }
    else {
        pushdown(p);
        int mid = (t[p].l + t[p].r) >> 1;
        if (pos <= mid)
        update(lc, pos, v);
        else
        update(rc, pos, v);
        maintain(p);
    }
}
void query(int p, int L, int R) { //调用之前要设置: mn = inf; mx =-inf; tot = 0;
    if (L <= t[p].l && R >= t[p].r) {
        tot += t[p].sum;
        mn = min(mn, t[p].min);
        mx = max(mx, t[p].max);
    }
    else {
        pushdown(p);
        int mid = (t[p].l + t[p].r) >> 1;
        if (L <= mid)
        query(lc, L, R);
        if (R > mid)
        query(rc, L, R);
    }
}

int main() {
    default_random_engine e;
    int n = 100000, m = 100000;
    uniform_int_distribution<int> d(-n, n);
    for (int i =-n; i <= n; ++i)
    A[i] = d(e);
    init();
    int root = build(-n, n);
    for (int i = 1; i <= m; ++i) {
        int op = rand() % 4, a = d(e), b = d(e), v = rand();
        int L = min(a, b), R = max(a, b);
        if (op == 0) {
            for (int i = L; i <= R; ++i)
            A[i] += v;
            update(root, L, R, op, v);
        }
        else if (op == 1) {
            for (int i = L; i <= R; ++i)
            A[i] = v;
            update(root, L, R, op, v);
        }
        else if (op == 2) {
            mn = inf; mx =-inf; tot = 0;
            query(root, L, R);
            if (mn != *min_element(A.data + offset + L, A.data + offset + R + 1))
            abort();
            if (mx != *max_element(A.data + offset + L, A.data + offset + R + 1))
            abort();
            if (tot != accumulate(A.data + offset + L, A.data + offset + R + 1,
0))
            abort();
        }
        else {
            A[L] += v;
            update(root, L, v);
        }
    }
    return 0;
}
```

逆元

```
const long long maxn = 1000005, mod = 1000000007;
long long pow(long long a, long long n, long long p) {
    long long ans = 1;
    while (n) {
        if (n & 1)
        ans = ans * a % p;
        a = a * a % p;
        n >>= 1;
    }
    return ans;
}
long long inverse1(long long a, long long n) { //费马小定理求逆元
    return pow(a, n- 2, n);
}
void extgcd(long long a, long long b, long long& d, long long& x, long long& y) {
    if (!b) { d = a; x = 1; y = 0; }
    else { extgcd(b, a % b, d, y, x); y-= x * (a / b); }
}
long long inverse2(long long a, long long n) {
    long long d, x, y;
    extgcd(a, n, d, x, y);
    return d == 1 ? (x + n) % n :-1;
}
long long inv[maxn];
void inverse3(long long n, long long p) {
    inv[1] = 1;
    for (long long i = 2; i <= n; ++i)
    inv[i] = (p- p / i) * inv[p % i] % p;
}
int main() {
    int number = 888;
    inverse3(100005, mod);
    printf("%lld %lld %lld\n", inverse1(number, mod), inverse2(number, mod),
inv[number]);
    return 0;
}
```

欧拉函数

```
//phi(n) 表示小于 n 且与 n 互素的整数个数
const int maxn = 1000000;
int vis[maxn], prime[maxn], phi[maxn], cnt;
void init() {
    memset(vis, 0, sizeof(vis));
    phi[1] = 1;
    cnt = 0;
    for (int i = 2; i < maxn; i++) {
        if (!vis[i]) {
            prime[cnt++] = i;
            phi[i] = i- 1;
        }
        for (int j = 0; j < cnt && i * prime[j] < maxn; j++) {
            int t = i * prime[j];
            vis[t] = 1;
            if (i % prime[j] == 0) {
                phi[t] = phi[i] * prime[j];
                break;
            }
            else {
                phi[t] = phi[i] * phi[prime[j]];
            }
        }
    }
}
int euler(int n) { //时间复杂度 O(sqrt(n))
    int ans = n;
    for (int i = 2; i * i <= n; ++i) if (n % i == 0) {
        ans = ans / i * (i- 1);
        while (n % i == 0)
        n /= i;
    }
    if (n > 1)
    ans = ans / n * (n- 1);
    return ans;
}
int main() {
    init();
    for (int i = 0; i <= 10000; ++i) if (phi[i] != euler(i))
    printf("%d\n", i);
    return 0;
}
```

线性筛素数

```
const int maxn = 1000000;
int vis[maxn], prime[maxn], cnt;
void init() {
        memset(vis, 0, sizeof(vis));
        cnt = 0;
        for (int i = 2; i < maxn; i++) {
        if (!vis[i])
        prime[cnt++] = i;
        for (int j = 0; j < cnt && i * prime[j] < maxn; j++) {
            int t = i * prime[j];
            vis[t] = 1;
            if (i % prime[j] == 0)
            break;
        }
    }
}
int main() {
    init();
    for (int i = 0; i < 10; ++i)
    printf("%d\n", prime[i]);
    return 0;
}
```

Dinic网络最大流/最小割

```
const int maxn = 100000 + 10;
const int inf = 1 << 30;
struct edge{
    int from, to, cap, flow;
    edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
};
struct Dinic {
    int n, m, s, t;
    vector<edge> edges;
    // 边数的两倍
    vector<int> G[maxn]; // 邻接表，G[i][j] 表示结点 i 的第 j 条边在 e 数组中的序号
    bool vis[maxn];
    // BFS 使用
    int d[maxn];
    int cur[maxn];
    void init(int n) {
        this->n = n;
        // 从起点到 i 的距离
        // 当前弧指针
        for (int i = 0; i < n; i++)
        G[i].clear();
        edges.clear();
    }
    void clear() {
        for (int i = 0; i < edges.size(); i++)
        edges[i].flow = 0;
    }
    void reduce() {
        for (int i = 0; i < edges.size(); i++)
        edges[i].cap-= edges[i].flow;
    }
    void addedge(int from, int to, int cap) {
        edges.push_back(edge(from, to, cap, 0));
        edges.push_back(edge(to, from, 0, 0));
        m = edges.size();
        G[from].push_back(m- 2);
        G[to].push_back(m- 1);
    }
    bool BFS() {
        memset(vis, 0, sizeof(vis));
        queue<int> Q;
        Q.push(s);
        vis[s] = 1;
        d[s] = 0;
        while (!Q.empty()) {
            int x = Q.front(); Q.pop();
            for (int i = 0; i < G[x].size(); i++) {
                edge& e = edges[G[x][i]];
                if (!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = 1;
                    d[e.to] = d[x] + 1;
                    Q.push(e.to);
                }
            }
        }
        return vis[t];
    }
    int DFS(int x, int a) {
        if (x == t || a == 0) return a;
        int flow = 0, f;
        for (int& i = cur[x]; i < G[x].size(); i++) {
            edge& e = edges[G[x][i]];
            if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap- e.flow))) > 0)
{
                e.flow += f;
                edges[G[x][i] ^ 1].flow-= f;
                flow += f;
                a-= f;
                if (a == 0) break;
            }
        }
        return flow;
    }
    int Maxflow(int s, int t) {
        this->s = s; this->t = t;
        int flow = 0;
        while (BFS()) {
            memset(cur, 0, sizeof(cur));
            flow += DFS(s, inf);
        }
        return flow;
    }
    vector<int> Mincut() { // call this after maxflow
        vector<int> ans;
            for (int i = 0; i < edges.size(); i++) {
            edge& e = edges[i];
            if (vis[e.from] && !vis[e.to] && e.cap > 0)
            ans.push_back(i);
        }
        return ans;
    }
}dinic;
int main() {
    freopen("D:\\in.txt", "r", stdin);
    int n, m;
    scanf("%d %d", &n, &m);
    dinic.init(n + 5);
    while (m--) {
        int s, t, u;
        scanf("%d %d %d", &s, &t, &u);
        dinic.addedge(s, t, u);
    }
    auto start = clock();
    printf("%d\n", dinic.Maxflow(1, n));
    double tot = static_cast<double>(clock()- start) / CLOCKS_PER_SEC;
    printf("Dinic: %f\n", tot);
    return 0;
}
```

KMP 算法

```
char T[] = "abcdefabc", P[] = "abc";
const int maxn = 10000;
int f[maxn];
//f[i] 表示字符串 s[0, i-1] 的后缀与前缀的最长公共部分（后缀与前缀均不包含字符串本身）
//若 f[i] = k 则，字符串 s[0, k-1] 与字符串 s[i-k, i-1] 相同
void getfail(char* P, int* f) {
    int m = strlen(P);
    f[0] = 0; f[1] = 0;
    for (int i = 1; i < m; ++i) {
        int j = f[i];
        while (j && P[j] != P[i])
        j = f[j];
        f[i + 1] = P[j] == P[i] ? j + 1 : 0;
    }
}
void find(char* T, char* P, int* f) {
    int n = strlen(T), m = strlen(P);
    //getfail(P, f);
    int j = 0;
    for (int i = 0; i < n; ++i) {
        while (j && P[j] != T[i])
        j = f[j];
        if (P[j] == T[i])
        ++j;
        if (j == m)
        printf("%d\n", i- m + 1); //在串 T 中找到了 P, 下标为 i- m + 1
    }
}
int main() {
    getfail(P, f);
    find(T, P, f);
    return 0;
}
```