# 杜教筛

```cpp
#include <cstring>
#include <iostream>
#include <map>
using namespace std;
constexpr int MAXN = 2000010;
using i64 = long long;
i64 T, n, pri[MAXN], cur, mu[MAXN], sum_mu[MAXN];
bool vis[MAXN];
map<i64, i64> mp_mu;

i64 S_mu(i64 x) {  // 求mu的前缀和
  if (x < MAXN) return sum_mu[x];
  if (mp_mu[x]) return mp_mu[x];  // 如果map中已有该大小的mu值，则可直接返回
  i64 ret = (i64)1;
  for (i64 i = 2, j; i <= x; i = j + 1) {
    j = x / (x / i);
    ret -= S_mu(x / i) * (j - i + 1);
  }
  return mp_mu[x] = ret;  // 路径压缩，方便下次计算
}

i64 S_phi(i64 x) {  // 求phi的前缀和
  i64 ret = (i64)0;
  i64 j;
  for (i64 i = 1; i <= x; i = j + 1) {
    j = x / (x / i);
    ret += (S_mu(j) - S_mu(i - 1)) * (x / i) * (x / i);
  }
  return (ret - 1) / 2 + 1;
}

signed main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> T;
  mu[1] = 1;
  for (int i = 2; i < MAXN; i++) {  // 线性筛预处理mu数组
    if (!vis[i]) {
      pri[++cur] = i;
      mu[i] = -1;
    }
    for (int j = 1; j <= cur && i * pri[j] < MAXN; j++) {
      vis[i * pri[j]] = true;
      if (i % pri[j])
        mu[i * pri[j]] = -mu[i];
      else {
        mu[i * pri[j]] = 0;
        break;
      }
    }
  }
  for (int i = 1; i < MAXN; i++)
    sum_mu[i] = sum_mu[i - 1] + mu[i];  // 求mu数组前缀和
```

```
  while (T--) {
    cin >> n;
    cout << S_phi(n) << ' ' << S_mu(n) << '\n';
  }
  return 0;
}
```

## 2—SAT—Tarjan

```
struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + f].push_back(2 * v + g);
    }
    bool satisfiable() {
        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        std::vector<int> stk;
        int now = 0, cnt = 0;
        std::function<void(int)> tarjan = [&](int u) {
            stk.push_back(u);
            dfn[u] = low[u] = now++;
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    tarjan(v);
                    low[u] = std::min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = std::min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    id[v] = cnt;
                } while (v != u);
                ++cnt;
            }
        };
        for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
        for (int i = 0; i < n; ++i) {
            if (id[2 * i] == id[2 * i + 1]) return false;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        return true;
    }
    std::vector<bool> answer() { return ans; }
};
```

# SCC Tarjan

```cpp
struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    SCC() {}
    SCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n + 1, {});
        dfn.assign(n + 1, -1);
        low.resize(n + 1);
        bel.assign(n + 1, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }

    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            ++cnt;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
        }
    }

    std::vector<int> work() {
        for (int i = 1; i <= n; i++) {
            if (dfn[i] == -1) {
```

```
                dfs(i);
            }
        }
        return bel;
    }
};
```

## 割点

```
struct CutPoint {
    int n, m, idx;
    std::vector<int> dfn, low, vis, cut;
    std::vector<std::vector<int>> adj;
    CutPoint(int _n, int _m) : n(_n), m(_m), dfn(_n + 1),
    low(_n + 1), vis(_n + 1), cut(_n + 1), adj(_n + 1) {

    }

    void dfs(int x, int root) {
        vis[x] = 1;
        dfn[x] = ++idx;
        low[x] = idx;
        int child = 0;
        for (auto y : adj[x]) {
            if (!vis[y]) {
                dfs(y, root);
                low[x] = std::min(low[x], low[y]);
                if (low[y] >= dfn[x] && x != root) {
                    cut[x] = 1;
                }
                if (x == root) {
                    child++;
                }
            }
            low[x] = std::min(low[x], dfn[y]);
        }
        if (child >= 2 && x == root) {
            cut[x] = 1;
        }
    }

    std::vector<int> work() {
        std::vector<int> q;
        for (int i = 1; i <= n; i++) {
            if (!vis[i]) {
                dfs(i, i);
            }
        }
        for (int i = 1; i <= n; i++) {
            if (cut[i]) {
                q.push_back(i);
            }
        }
```

```
        return q;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }
};
```

## 割边

```
struct CutEdges {
    int n;
    int idx = 0;
    vector<int> low, dfn, fa;
    vector<int> head, nxt, to;
    vector<int> b;
    int iddx = 1;
    vector<pair<int,int>> bridge;
    CutEdges(int n, int m) : low(n + 1), dfn(n + 1), fa(n + 1),
    head(n + 1), to(2 * m + 4), nxt(2 * m + 4), b(2 * m + 4) {
        this->n = n;
    }
    void addEdge(int x, int y) {
        nxt[++iddx] = head[x];
        head[x] = iddx;
        to[iddx] = y;
    }
    vector<pair<int, int>> work() {
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) tarjan(i, 0);
        }
        return bridge;
    }
    void tarjan(int x, int e_in) {;
        dfn[x] = low[x] = ++idx;
        for(int i = head[x]; i; i = nxt[i]) {
            int y = to[i];
            if(!dfn[y]) {
                tarjan(y, i);
                if(dfn[x] < low[y]) {
                    bridge.push_back({x, y});
                    b[i] = b[i ^ 1] = 1;
                }
                low[x] = min(low[x], low[y]);
            } else if (i != (e_in ^ 1)) {
                low[x] = min(low[x], dfn[y]);
            }
        }
    }

};
CutEdges g(n, m);
```