Powerful Number 筛

定义

Powerful Number（以下简称 PN）筛类似于杜教筛，或者说是杜教筛的一个扩展，可以拿来求一些积性函数的前缀和。

要求：

- 存在一个函数 g 满足：
  - g 是积性函数。
  - g 易求前缀和。
  - 对于质数 p，$g(p) = f(p)$。

假设现在要求积性函数 f 的前缀和 $F(n) = \sum_{i=1}^{n} f(i)$。

Powerful Number

定义：对于正整数 n，记 n 的质因数分解为 $n = \prod_{i=1}^{m} p_i^{e_i}$。$n$ 是 PN 当且仅当 $\forall 1 \le i \le m, e_i > 1$。

性质 1：所有 PN 都可以表示成 a^2b^3 的形式。

证明：若 e_i 是偶数，则将 $p_i^{e_i}$ 合并进 a^2 里；若 e_i 为奇数，则先将 p_i^3 合并进 b^3 里，再将 $p_i^{e_i-3}$ 合并进 a^2 里。

性质 2：n 以内的 PN 至多有 $O(\sqrt{n})$ 个。

证明：考虑枚举 a，再考虑满足条件的 b 的个数，有 PN 的个数约等于

$$\int_1^{\sqrt{n}} \sqrt[3]{\frac{n}{x^2}} \mathrm{d}x = O(\sqrt{n})$$

那么如何求出 n 以内所有的 PN 呢？线性筛找出 $\sqrt{n}$ 内的所有素数，再 DFS 搜索各素数的指数即可。由于 n 以内的 PN 至多有 $O(\sqrt{n})$ 个，所以至多搜索 $O(\sqrt{n})$ 次。

PN 筛

首先，构造出一个易求前缀和的积性函数 g，且满足对于素数 p，$g(p) = f(p)$。记 $G(n) = \sum_{i=1}^{n} g(i)$。

然后，构造函数 $h = f/g$，这里的 / 表示狄利克雷卷积除法。根据狄利克雷卷积的性质可以得知 h 也为积性函数，因此 $h(1) = 1$。$f = g * h$，这里 $*$ 表示狄利克雷卷积。

对于素数 p，$f(p) = g(1)h(p) + g(p)h(1) = h(p) + g(p) \implies h(p) = 0$。根据 $h(p) = 0$ 和 h 是积性函数可以推出对于非 PN 的数 n 有 $h(n) = 0$，即 h 仅在 PN 处取有效值。

现在，根据 $f = g * h$ 有

$$
\begin{aligned}
F(n) &= \sum_{i=1}^{n} f(i) \\
&= \sum_{i=1}^{n} \sum_{d|i} h(d) g\left(\frac{i}{d}\right) \\
&= \sum_{d=1}^{n} \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} h(d) g(i) \\
&= \sum_{d=1}^{n} h(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} g(i) \\
&= \sum_{d=1}^{n} h(d) G\left(\left\lfloor \frac{n}{d} \right\rfloor\right) \\
&= \sum_{\substack{d=1 \\ d \text{ is PN}}}^{n} h(d) G\left(\left\lfloor \frac{n}{d} \right\rfloor\right)
\end{aligned}
$$

$O(\sqrt{n})$ 找出所有 PN，计算出所有 h 的有效值。对于 h 有效值的计算，只需要计算出所有 $h(p^c)$ 处的值，就可以根据 h 为积性函数推出 h 的所有有效值。现在对于每一个有效值 d，计算 $h(d)G\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$ 并累加即可得到 $F(n)$。

下面考虑计算 $h(p^c)$，一共有两种方法：一种是直接推出 $h(p^c)$ 仅与 p, c 有关的计算公式，再根据公式计算 $h(p^c)$；另一种是根据 $f = g * h$ 有 $f(p^c) = \sum_{i=0}^{c} g(p^i) h(p^{c-i})$，移项可得 $h(p^c) = f(p^c) - \sum_{i=1}^{c} g(p^i) h(p^{c-i})$，现在就可以枚举素数 p 再枚举指数 c 求解出所有 $h(p^c)$。

过程

1. 构造 g
2. 构造快速计算 G 的方法
3. 计算 $h(p^c)$
4. 搜索 PN，过程中累加答案
5. 得到结果

对于第 3 步，可以直接根据公式计算，可以使用枚举法预处理打表，也可以搜索到了再临时推。

性质

以使用第二种方法计算 $h(p^c)$ 为例进行分析。可以分为计算 $h(p^c)$ 和搜索两部分进行分析。

对于第一部分，根据 $O(\sqrt{n})$ 内的素数个数为 $O\left(\frac{\sqrt{n}}{\log n}\right)$，每个素数 p 的指数 c 至多为 $\log n$，计算 $h(p^c)$ 需要循环 $(c-1)$ 次，由此有第一部分的时间复杂度为 $O\left(\frac{\sqrt{n}}{\log n}\cdot\log n\cdot\log n\right)=O(\sqrt{n}\log n)$，且这是一个宽松的上界。根据题目的不同还可以添加不同的优化，从而降低第一部分的时间复杂度。

对于搜索部分，由于 n 以内的 PN 至多有 $O(\sqrt{n})$ 个，所以至多搜索 $O(\sqrt{n})$ 次。对于每一个 PN，根据计算 G 的方法不同，时间复杂度也不同。例如，假设计算 $G\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$ 的时间复杂度为 $O(1)$，则第二部分的复杂度为 $O(\sqrt{n})$。

特别地，若借助杜教筛计算 $G\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$，则第二部分的时间复杂度为杜教筛的时间复杂度，即 $O(n^{\frac{2}{3}})$。因为若事先计算一次 $G(n)$，并且预先使用线性筛优化和用支持快速随机访问的数据结构（如 C++ 中的 `std::map` 和 `std::unordered_map`）记录较大的值，则杜教筛过程中用到的 $G\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$ 都是线性筛中记录的或者 `std::map` 中记录的，这一点可以直接用程序验证。

对于空间复杂度，其瓶颈在于存储 $h(p^c)$。若使用二维数组 a 记录，$a_{i,j}$ 表示 $h(p_i^j)$ 的值，则空间复杂度为 $O\left(\frac{\sqrt{n}}{\log n}\cdot\log n\right)=O(\sqrt{n})$。

例题

Luogu P5325【模板】Min_25 筛

题意：给定积性函数 $f(p^k)=p^k(p^k-1)$，求 $\sum_{i=1}^{n}f(i)$。

易得 $f(p)=p(p-1)=\mathrm{id}(p)\varphi(p)$，构造 $g(n)=\mathrm{id}(n)\varphi(n)$。

考虑使用杜教筛求 $G(n)$，根据 $(\mathrm{id}\cdot\varphi)*\mathrm{id}=\mathrm{id}_2$ 可得 $G(n)=\sum_{i=1}^{n}i^2-\sum_{d=2}^{n}d\cdot G\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$。

之后 $h(p^k)$ 的取值可以枚举计算，这种方法不再赘述。

此外，此题还可以直接求出 $h(p^k)$ 仅与 p,k 有关的公式，过程如下：

$$
\begin{aligned}
&f(p^k)=\sum_{i=0}^{k} g(p^{k-i})h(p^i)\\
\iff &p^k(p^k-1)=\sum_{i=0}^{k} p^{k-i}\varphi(p^{k-i})h(p^i)\\
\iff &p^k(p^k-1)=\sum_{i=0}^{k} p^{2k-2i-1}(p-1)h(p^i)\\
\iff &p^k(p^k-1)=h(p^k)+\sum_{i=0}^{k-1} p^{2k-2i-1}(p-1)h(p^i)\\
\iff &h(p^k)=p^k(p^k-1)-\sum_{i=0}^{k-1} p^{2k-2i-1}(p-1)h(p^i)\\
\iff &h(p^k)-p^2h(p^{k-1})=p^k(p^k-1)-p^{k+1}(p^{k-1}-1)-p(p-1)h(p^{k-1})\\
\iff &h(p^k)-ph(p^{k-1})=p^{k+1}-p^k\\
\iff &\frac{h(p^k)}{p^k}-\frac{h(p^{k-1})}{p^{k-1}}=p-1
\end{aligned}
$$

再根据 $h(p)=0$，通过累加法即可推出 $h(p^k)=(k-1)(p-1)p^k$。

参考代码

```cpp
#include <iostream>
#include <map>
using namespace std;

constexpr int MOD = 1e9 + 7;

template <typename T>
int mint(T x) {
  x %= MOD;
  if (x < 0) x += MOD;
  return x;
}

int add(int x, int y) { return x + y >= MOD ? x + y - MOD : x +
y; }

int mul(int x, int y) { return (long long)1 * x * y % MOD; }

int sub(int x, int y) {
  return x < y ? x - y + MOD : x - y;  // 防止负数
}

int qp(int x, int y) {
  int r = 1;
  for (; y; y >>= 1) {
    if (y & 1) r = mul(r, x);
    x = mul(x, x);
  }
  return r;
}

int inv(int x) { return qp(x, MOD - 2); }

namespace PNS {
constexpr int N = 2e6 + 5;
constexpr int M = 35;

long long global_n;

int g[N], sg[N];

int h[N][M];
bool vis_h[N][M];

int ans;

int pcnt, prime[N], phi[N];
bool isp[N];

```

```
void sieve(int n) {
  pcnt = 0;
  for (int i = 2; i <= n; ++i) isp[i] = true;  // 判断质数数组
  phi[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (isp[i]) {
      ++pcnt;
      prime[pcnt] = i;
      phi[i] = i - 1;
    }
    for (int j = 1; j <= pcnt; ++j) {  // 筛去非质数
      long long nxt = (long long)1 * i * prime[j];
      if (nxt > n) break;
      isp[nxt] = false;
      if (i % prime[j] == 0) {  // i是非质数的情况
        phi[nxt] = phi[i] * prime[j];
        break;
      }
      phi[nxt] = phi[i] * phi[prime[j]];
    }
  }

  for (int i = 1; i <= n; ++i) g[i] = mul(i, phi[i]);

  sg[0] = 0;
  for (int i = 1; i <= n; ++i) sg[i] = add(sg[i - 1], g[i]);  //
g函数的前缀和
}

int inv2, inv6;

void init() {
  sieve(N - 1);
  for (int i = 1; i <= pcnt; ++i) h[i][0] = 1, h[i][1] = 0;
  for (int i = 1; i <= pcnt; ++i) vis_h[i][0] = vis_h[i][1] =
true;
  inv2 = inv(2);
  inv6 = inv(6);
}

int S1(long long n) { return mul(mul(mint(n), mint(n + 1)),
inv2); }

int S2(long long n) {
  return mul(mul(mint(n), mul(mint(n + 1), mint(n * 2 + 1))),
inv6);
}

map<long long, int> mp_g;

int G(long long n) {
  if (n < N) return sg[n];
```

```
  if (mp_g.count(n)) return mp_g[n];

  int ret = S2(n);
  for (long long i = 2, j; i <= n; i = j + 1) {
    j = n / (n / i);
    ret = sub(ret, mul(sub(S1(j), S1(i - 1)), G(n / i)));
  }
  mp_g[n] = ret;
  return ret;
}

void dfs(long long d, int hd, int pid) {
  ans = add(ans, mul(hd, G(global_n / d)));

  for (int i = pid; i <= pcnt; ++i) {
    if (i > 1 && d > global_n / prime[i] / prime[i]) break;  //
剪枝

    int c = 2;
    for (long long x = d * prime[i] * prime[i]; x <= global_n;
         x *= prime[i], ++c) {  // 计算f.g函数
      if (!vis_h[i][c]) {
        int f = qp(prime[i], c);
        f = mul(f, sub(f, 1));
        int g = mul(prime[i], prime[i] - 1);
        int t = mul(prime[i], prime[i]);

        for (int j = 1; j <= c; ++j) {
          f = sub(f, mul(g, h[i][c - j]));
          g = mul(g, t);
        }
        h[i][c] = f;
        vis_h[i][c] = true;
      }

      if (h[i][c]) dfs(x, mul(hd, h[i][c]), i + 1);
    }
  }
}

int solve(long long n) {
  global_n = n;
  ans = 0;
  dfs(1, 1, 1);
  return ans;
}
}  // namespace PNS

int main() {
  PNS::init();
  long long n;
  cin >> n;
```

```
    cout << PNS::solve(n) << '\n';
    return 0;
}
```

「LOJ #6053」简单的函数

给定 $f(n)$：

$$f(n)=\begin{cases}1 & n=1\\ p\oplus c & n=p^c\\ f(a)f(b) & n=ab\text{ and }a\perp b\end{cases}$$

易得：

$$f(p)=\begin{cases}p+1 & p=2\\ p-1 & \text{otherwise}\end{cases}$$

构造 g 为

$$g(n)=\begin{cases}3\varphi(n) & 2\mid n\\ \varphi(n) & \text{otherwise}\end{cases}$$

易证 $g(p)=f(p)$ 且 g 为积性函数。

下面考虑求 $G(n)$。

$$\begin{aligned}G(n)&=\sum_{i=1}^{n}[i\bmod 2=1]\varphi(i)+3\sum_{i=1}^{n}[i\bmod 2=0]\varphi(i)\\&=\sum_{i=1}^{n}\varphi(i)+2\sum_{i=1}^{n}[i\bmod 2=0]\varphi(i)\\&=\sum_{i=1}^{n}\varphi(i)+2\sum_{i=1}^{\lfloor\frac{n}{2}\rfloor}\varphi(2i)\end{aligned}$$

记 $S_1(n)=\sum_{i=1}^{n}\varphi(i)$，$S_2(n)=\sum_{i=1}^{n}\varphi(2i)$，则 $G(n)=S_1(n)+2S_2\left(\left\lfloor\frac{n}{2}\right\rfloor\right)$。

当 $2\mid n$ 时，有

$$\begin{aligned}S_2(n)&=\sum_{i=1}^{n}\varphi(2i)\\&=\sum_{i=1}^{\frac{n}{2}}(\varphi(2(2i-1))+\varphi(2(2i)))\\&=\sum_{i=1}^{\frac{n}{2}}(\varphi(2i-1)+2\varphi(2i))\\&=\sum_{i=1}^{\frac{n}{2}}(\varphi(2i-1)+\varphi(2i))+\sum_{i=1}^{\frac{n}{2}}\varphi(2i)\\&=\sum_{i=1}^{n}\varphi(i)+S_2\left(\frac{n}{2}\right)\\&=S_1(n)+S_2\left(\left\lfloor\frac{n}{2}\right\rfloor\right)\end{aligned}$$

当 $2 \nmid n$ 时，有

$$
\begin{aligned}
S_2(n) &= S_2(n-1) + \varphi(2n) \\
&= S_2(n-1) + \varphi(n) \\
&= \sum_{i=1}^{n-1} \varphi(i) + S_2\left(\frac{n-1}{2}\right) + \varphi(n) \\
&= S_1(n) + S_2\left(\left\lfloor \frac{n}{2} \right\rfloor\right)
\end{aligned}
$$

综上，有 $S_2(n) = S_1(n) + S_2\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$。

S_1 可以用杜教筛求，S_2 直接按照公式推，这样 G 也可以求出来了。

参考代码

```cpp
#include <iostream>
#include <map>
using namespace std;

constexpr int MOD = 1e9 + 7;
constexpr int inv2 = (MOD + 1) / 2;

template <typename T>
int mint(T x) {
  x %= MOD;
  if (x < 0) x += MOD;
  return x;
}

int add(int x, int y) {
  return x + y >= MOD ? x + y - MOD : x + y;  // 防止大于模数
}

int mul(int x, int y) { return (long long)1 * x * y % MOD; }

int sub(int x, int y) {
  return x < y ? x - y + MOD : x - y;  // 防负数
}

namespace PNS {
constexpr int N = 2e6 + 5;
constexpr int M = 35;

long long global_n;

int s1[N], s2[N];

int h[N][M];
bool vis_h[N][M];

int ans;

int pcnt, prime[N], phi[N];
bool isp[N];

void sieve(int n) {
  pcnt = 0;
  for (int i = 2; i <= n; ++i) isp[i] = true;  // 判断质数数组
  phi[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (isp[i]) {
      ++pcnt;
      prime[pcnt] = i;
      phi[i] = i - 1;
```

```
    }
    for (int j = 1; j <= pcnt; ++j) {  // 筛去非质数
      long long nxt = (long long)1 * i * prime[j];
      if (nxt > n) break;
      isp[nxt] = false;
      if (i % prime[j] == 0) {  // i是非质数的情况
        phi[nxt] = phi[i] * prime[j];
        break;
      }
      phi[nxt] = phi[i] * phi[prime[j]];
    }
  }

  s1[0] = 0;
  for (int i = 1; i <= n; ++i) s1[i] = add(s1[i - 1], phi[i]);

  s2[0] = 0;
  for (int i = 1; i <= n / 2; ++i) {
    s2[i] = add(s2[i - 1], phi[2 * i]);
  }
}

void init() {
  sieve(N - 1);
  for (int i = 1; i <= pcnt; ++i) h[i][0] = 1;
  for (int i = 1; i <= pcnt; ++i) vis_h[i][0] = true;
}

map<long long, int> mp_s1;

int S1(long long n) {
  if (n < N) return s1[n];
  if (mp_s1.count(n)) return mp_s1[n];

  int ret = mul(mul(mint(n), mint(n + 1)), inv2);
  for (long long i = 2, j; i <= n; i = j + 1) {
    j = n / (n / i);
    ret = sub(ret, mul(mint(j - i + 1), S1(n / i)));
  }
  mp_s1[n] = ret;
  return ret;
}

map<long long, int> mp_s2;

int S2(long long n) {
  if (n < N / 2) return s2[n];
  if (mp_s2.count(n)) return mp_s2[n];
  int ret = add(S1(n), S2(n / 2));
  mp_s2[n] = ret;
  return ret;
}
```

```cpp

int G(long long n) { return add(S1(n), mul(2, S2(n / 2))); }

void dfs(long long d, int hd, int pid) {
  ans = add(ans, mul(hd, G(global_n / d)));

  for (int i = pid; i <= pcnt; ++i) {
    if (i > 1 && d > global_n / prime[i] / prime[i]) break;  // 剪枝

    int c = 2;
    for (long long x = d * prime[i] * prime[i]; x <= global_n;
         x *= prime[i], ++c) {
      if (!vis_h[i][c]) {
        int f = prime[i] ^ c, g = prime[i] - 1;

        // p = 2时特判一下
        if (i == 1) g = mul(g, 3);

        for (int j = 1; j <= c; ++j) {
          f = sub(f, mul(g, h[i][c - j]));
          g = mul(g, prime[i]);
        }
        h[i][c] = f;
        vis_h[i][c] = true;
      }

      if (h[i][c]) dfs(x, mul(hd, h[i][c]), i + 1);
    }
  }
}

int solve(long long n) {
  global_n = n;
  ans = 0;
  dfs(1, 1, 1);
  return ans;
}
}  // namespace PNS

int main() {
  PNS::init();  // 预处理函数
  long long n;
  cin >> n;
  cout << PNS::solve(n) << '\n';
  return 0;
}
```

习题

- PE708 Twos are all you need
- PE639 Summing a multiplicative function
- PE484 Arithmetic Derivative

参考资料

- 破壁人五号 - Powerful number 筛略解
- command_block - 杜教筛（+ 贝尔级数 + powerful number）
