算法笔记

数据结构

并查集

使用 `std::unordered_map` 实现。

```
template<typename T>
class UQset {
private:
    unordered_map<T, T> mp;
    const T& get(const T& x) {
        if (!mp.count(x)) return x;
        T ans = x;
        stack<T> stk;
        while (mp.count(ans) && ans != mp[ans]) {
            stk.push(ans);
            ans = mp[ans];
        }
        // 每次查询时将查询路径上的所有结点连接到根上
        while (!stk.empty()) {
            mp[stk.top()] = ans;
            stk.pop();
        }
        return ans;
    }
public:
    // 将x和y所在集合合并
    void U(const T& x, const T& y) { mp[get(y)] = get(x); }
    // 查询x和y是否在同一集合内
    bool Q(const T& x, const T& y) { return get(x) == get(y); }
};
```

对于 `int` 类型的简化：

```
class UQset {
private:
    unordered_map<int, int> mp;
    int get(int x) {
        if (!mp.count(x)) return x;
        int ans = x;
        stack<int> stk;
        while (mp.count(ans) && ans != mp[ans]) {
            stk.push(ans);
```

```
            ans = mp[ans];
        }
        while (!stk.empty()) {
            mp[stk.top()] = ans;
            stk.pop();
        }
        return ans;
    }
public:
    void U(int x, int y) { mp[get(y)] = get(x); }
    bool Q(int x, int y) { return get(x) == get(y); }
};
```

算法

快速幂

基本原理:

$$a^b = \begin{cases} a \cdot a^{b-1}, & b\text{为奇数} \\ \left(a^{\frac{b}{2}}\right)^2, & b\text{为偶数}. \end{cases}$$

例如求2^{13}的过程如下:

1. $a^{13} = a \times a^{12}$;
2. $a^{12} = \left(a^6\right)^2$;
3. $a^6 = \left(a^3\right)^2$;
4. $a^3 = a \times a^2$;
5. $a^2 = \left(a\right)^2$;
6. $a = a \times 1$.

算法时间复杂度为$O(\log n)$。

递归实现

```
// 计算 a^b mod m, m为大质数
int binaryPow(int a, int b, int m) {
        if (b == 0) return 1;  // a^0 = 1
        if (b & 1) {  // b & 1 等价于 b % 2 == 1, 判断b的奇偶性
                return a * binaryPow(a, b - 1, m) % m;
        } else {
                int mul = binaryPow(a, b / 2, m);
                return mul * mul % m;
```

```
        }
}
```

注意如果 `b` 为偶数时直接 `return binaryPow(a, b/2, m) * binaryPow(a, b/2, m)` 这样时间复杂度仍然是$O(n)$。
考虑上面a^{13}的例子，可以将任意正整数分解为一系列2的幂之和且分解唯一，如 $13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0 = 1101_{(2)}$，因此$a^{13} = a^8 \times a^4 \times a^1$。

迭代实现

```
// 计算 a^b mod m，m为大质数
int binaryPow(int a, int b, int m) {
        int ans = 1;
        while (b > 0) {
                if (b & 1) {  // b的二进制末尾为1
                        ans = ans * a % m;
                }
                a = a * a % m;
                b >>= 1;  // b右移一位
        }
        return ans;
}
```

素数

埃拉托斯特尼筛法

给定n，输出不大于n的所有质数。

```
vector<int> prime(int n) {
    vector<int> ans;  // 保存结果
    vector<int> flag(n + 1, 0);  // 标记数组，0为质数，1为合数。arr[0~1]不使用
    for (int i = 2; i < n + 1); ++i) {  // 从最小的质数2开始遍历
        if (!flag[i]) {  // i为质数
            ans.push_back(i);
            for (int j = i * 2; j < n + 1; j += i) flag[j] = 1;  // 标记i的所有倍数
        }
    }
    return ans;
}
```

如果只需要处理标记数组，最外层 `i` 的遍历为 `0 ~ sqrt(n)` 即可。

质因数分解

给定n，输出质因式分解的结果（质因数和对应的指数）。
思路：

1. 计算n的所有质因数（若n为合数，最大质因数为n/2；若n为质数，则最大质因数为n)；
2. 依次用n除以质因数，每次可以整除时累加指数数组一次。

```
vector<int> p = prime(n);  // 不大于n的所有质数
vector<int> a(p.size());  // 各质数对应的指数数组
for (int i = 0; i < p.size(); ++i) {
    while (n % p[i] == 0) {
        n /= p[i];
        ++a[i];
    }
}
```

求$n!$的质因数分解

基本思路：从$2 \sim n$依次求其质因数分解，累加指数。
优化思路：对于质数2，考虑$2 \sim n$中的数，所有偶数都包含1个质因数2，而所有4的倍数都额外包含1个质因数2，所有8的倍数又额外包含一个质因数2……因此对于$n!$，质因数p的指数为：

$$\left[\frac{n}{p}\right] + \left[\frac{n}{p^2}\right] + \left[\frac{n}{p^3}\right] + \left[\frac{n}{p^4}\right] + \cdots + \left[\frac{n}{p^k}\right]$$

其中$[\cdot]$为取整函数，直到$n < p^k$。

```
vector<int> p = prime(n);  // 不大于n的所有质数
vector<int> a(p.size());  // 各质数对应的指数数组
for (int i = 0; i < p.size(); ++i) {
    for (int wk = n; wk > 0;) {
        wk /= p[i];
        a[i] += wk;
    }
}
```