CSP 复赛复习 - 基础数学与数论

By 奇思妙学

各种进制转换

进制基本概念

常见进制：

- 二进制（Binary）：基数为 2，数字 $0-1$
- 八进制（Octal）：基数为 8，数字 $0-7$
- 十进制（Decimal）：基数为 10，数字 $0-9$
- 十六进制（Hexadecimal）：基数为 16，数字 $0-9, A-F$

转换公式： $N = a_n \times r^n + a_{n-1} \times r^{n-1} + \cdots + a_1 \times r^1 + a_0 \times r^0$

十进制转其他进制（除基取余法）

```cpp
// 十进制转二进制
void decimalToBinary(int n) {
    int binary[32], idx = 0;

    while (n > 0) {
        binary[idx++] = n % 2;
        n /= 2;
    }

    // 逆序输出
    for (int i = idx - 1; i >= 0; i--) {
        cout << binary[i];
    }
}
```

```cpp
// 十进制转任意进制 (2-16)
void decimalToBase(int n, int base) {
    char digits[] = "0123456789ABCDEF";
    char result[32];
    int idx = 0;

    while (n > 0) {
        result[idx++] = digits[n % base];
        n /= base;
    }

    // 逆序输出
    for (int i = idx - 1; i >= 0; i--) {
        cout << result[i];
    }
}
```

其他进制转十进制（按权展开法）

```cpp
// 二进制转十进制
int binaryToDecimal(string binary) {
    int decimal = 0;
    int power = 1;

    for (int i = binary.length() - 1; i >= 0; i--) {
        if (binary[i] == '1') {
            decimal += power;
        }
        power *= 2;
    }
    return decimal;
}
```

```cpp
// 任意进制转十进制 (2-16)
int baseToDecimal(string num, int base) {
    int decimal = 0;
    int power = 1;

    for (int i = num.length() - 1; i >= 0; i--) {
        char c = num[i];
        int digit;

        if (c >= '0' && c <= '9') {
            digit = c - '0';
        } else {
            digit = c - 'A' + 10;
        }

        decimal += digit * power;
        power *= base;
    }
    return decimal;
}
```

进制转换综合示例

```cpp
// 十进制转任意进制
string decimalToBase(int n, int base) {
    if (n == 0) return "0";

    string digits = "0123456789ABCDEF";
    string result = "";

    while (n > 0) {
        result += digits[n % base];
        n /= base;
    }

    reverse(result.begin(), result.end());
    return result;
}
```

```cpp
// 任意进制转十进制
int baseToDecimal(string num, int base) {
    int result = 0;
    int power = 1;

    for (int i = num.length() - 1; i >= 0; i--) {
        char c = num[i];
        int digit = (c >= '0' && c <= '9') ? (c - '0') : (c - 'A' + 10);
        result += digit * power;
        power *= base;
    }
    return result;
}

int main() {
    int n = 255;
    cout << decimalToBase(n, 2) << endl;  // 11111111
    cout << decimalToBase(n, 8) << endl;  // 377
    cout << decimalToBase(n, 16) << endl; // FF

    string hex = "FF";
    cout << baseToDecimal(hex, 16) << endl; // 255

    return 0;
}
```

加法原理和乘法原理

加法原理

定义：如果完成一项任务有 n 类方法，第 i 类方法有 a_i 种方式，且这些方法互不重叠，则完成该任务共有：

$$a_1 + a_2 + \cdots + a_n \text{ 种方式}$$

示例：从 3 本数学书和 4 本语文书中选一本书，有 $3 + 4 = 7$ 种选择。

乘法原理

定义：如果完成一项任务需要 n 个步骤，第 i 个步骤有 a_i 种方法，且各步骤相互独立，则完成该任务共有：

$$a_1 \times a_2 \times \cdots \times a_n \text{ 种方式}$$

示例：从 3 件上衣和 4 条裤子中选一套衣服，有 $3 \times 4 = 12$ 种搭配。

综合应用示例

```cpp
// 计算路径数量问题
int countPaths(int m, int n) {
    // 从左上角到右下角的路径数
    // 只能向右或向下移动
    return combination(m + n - 2, m - 1);
}

// 计算密码组合数
int countPasswords(int length, bool useDigits, bool useLetters) {
    int choices = 0;
    if (useDigits) choices += 10;  // 0-9
    if (useLetters) choices += 26; // a-z

    int total = 1;
    for (int i = 0; i < length; i++) {
        total *= choices;
    }
    return total;
}
```

排列与组合

排列（Permutation）

定义：从 n 个不同元素中取出 m 个元素按一定顺序排列：

$$P(n,m) = \frac{n!}{(n-m)!} = n \times (n-1) \times \cdots \times (n-m+1)$$

全排列：$P(n,n) = n!$

```cpp
// 计算排列数 P(n, m)
long long permutation(int n, int m) {
    long long result = 1;
    for (int i = 0; i < m; i++) {
        result *= (n - i);
    }
    return result;
}

// 生成全排列
void generatePermutations(int arr[], int start, int n) {
    if (start == n) {
        // 输出当前排列
        for (int i = 0; i < n; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
        return;
    }

    for (int i = start; i < n; i++) {
        swap(arr[start], arr[i]);
        generatePermutations(arr, start + 1, n);
        swap(arr[start], arr[i]); // 回溯
    }
}
```

组合（Combination）

定义：从 n 个不同元素中取出 m 个元素，不考虑顺序：

$$C(n,m)=\binom{n}{m}=\frac{n!}{m!(n-m)!}$$

性质：

- $C(n,m)=C(n,n-m)$
- $C(n,m)=C(n-1,m-1)+C(n-1,m)$（杨辉三角）

```cpp
// 计算组合数 C(n, m)
long long combination(int n, int m) {
    if (m > n) return 0;
    if (m > n - m) m = n - m; // 利用对称性

    long long result = 1;
    for (int i = 1; i <= m; i++) {
        result = result * (n - i + 1) / i;
    }
    return result;
}

// 预处理组合数表
const int MAXN = 1000;
long long C[MAXN][MAXN];

void initCombination() {
    for (int i = 0; i < MAXN; i++) {
        C[i][0] = C[i][i] = 1;
        for (int j = 1; j < i; j++) {
            C[i][j] = C[i-1][j-1] + C[i-1][j];
        }
    }
}
```

22.1 整除的基本知识

整除定义与性质

定义：如果存在整数 k 使得 $b = a \times k$，则称 a 整除 b，记作 $a \mid b$

性质：

1. 若 $a \mid b$ 且 $b \mid c$，则 $a \mid c$
2. 若 $a \mid b$ 且 $a \mid c$，则 $a \mid (b \pm c)$
3. 若 $a \mid b$，则 $a \mid bc$（c 为任意整数）

整除判断与相关计算

```cpp
// 判断整除
bool isDivisible(int a, int b) {
    return b != 0 && a % b == 0;
}

// 获取所有因子
void getFactors(int n, int factors[], int &count) {
    count = 0;
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            factors[count++] = i;
            if (i != n / i) {
                factors[count++] = n / i;
            }
        }
    }
}

// 判断完全数（所有真因子和等于自身）
bool isPerfectNumber(int n) {
    int sum = 0;
    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            sum += i;
        }
    }
    return sum == n;
}
```

22.2 质数与合数

质数判断

质数：大于 1 且只能被 1 和自身整除的自然数

```cpp
// 判断单个质数
bool isPrime(int n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;

    for (int i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return false;
    }
    return true;
}

// 获取质数因子分解
void primeFactorization(int n) {
    for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
            cout << i << " ";
            n /= i;
        }
    }
    if (n > 1) cout << n;
}
```

埃拉托斯特尼筛法（埃式筛）

时间复杂度： $O(n \log \log n)$

```cpp
const int MAXN = 1000000;
bool isPrime[MAXN];

void eratosthenesSieve(int n) {
    for (int i = 2; i <= n; i++)
        isPrime[i] = true;

    for (int i = 2; i * i <= n; i++)
        if (isPrime[i]) {
            for (int j = i * i; j <= n; j += i) {
                isPrime[j] = false;
            }
        }

    // 输出所有质数
    for (int i = 2; i <= n; i++)
        if (isPrime[i])
            cout << i << " ";
}
```

欧拉筛法（线性筛）

时间复杂度： $O(n)$

```cpp
const int MAXN = 1000000;
bool isPrime[MAXN];
int primes[MAXN], primeCount = 0;

void eulerSieve(int n) {
    for (int i = 2; i <= n; i++) {
        if (!isPrime[i])
            primes[primeCount++] = i;

        for (int j = 0; j < primeCount && i * primes[j] <= n; j++) {
            isPrime[i * primes[j]] = true;
            if (i % primes[j] == 0) break;
        }
    }

    // 输出所有质数
    for (int i = 0; i < primeCount; i++)
        cout << primes[i] << " ";
}
```

22.3 最大公约数与最小公倍数

最大公约数（GCD）

定义：能同时整除两个数的最大正整数

性质：

- $\gcd(a, b) = \gcd(b, a)$
- $\gcd(a, b) = \gcd(a, b - a)$
- $\gcd(a, 0) = |a|$

```cpp
// 辗转相除法（欧几里得算法）
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

// 递归实现
int gcd_recursive(int a, int b) {
    return b == 0 ? a : gcd_recursive(b, a % b);
}

// 更相减损术
int gcd_subtraction(int a, int b) {
    while (a != b) {
        if (a > b) a -= b;
        else b -= a;
    }
    return a;
}
```

最小公倍数（LCM）

定义：能同时被两个数整除的最小正整数

公式：$\operatorname{lcm}(a,b) = \frac{a \times b}{\gcd(a,b)}$

```cpp
// 计算最小公倍数
int lcm(int a, int b) {
    return a / gcd(a, b) * b;  // 先除后乘避免溢出
}

// 多个数的最小公倍数
int lcm_multiple(int arr[], int n) {
    int result = arr[0];
    for (int i = 1; i < n; i++) {
        result = lcm(result, arr[i]);
    }
    return result;
}

// 多个数的最大公约数
int gcd_multiple(int arr[], int n) {
    int result = arr[0];
    for (int i = 1; i < n; i++) {
        result = gcd(result, arr[i]);
    }
    return result;
}
```

22.4 算术基本定理

定理内容

算术基本定理：任何一个大于 1 的自然数，都可以唯一分解为质因数的乘积：

$$n = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k}$$

其中 $p_1, p_2, \ldots, p_k$ 是质数，$a_1, a_2, \ldots, a_k$ 是正整数。

质因数分解实现

```cpp
// 质因数分解
void primeFactorize(int n, map<int, int> &factors) {
    for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
            factors[i]++;
            n /= i;
        }
    }
    if (n > 1) {
        factors[n]++;
    }
}
```

```cpp
// 输出质因数分解结果
void printFactorization(int n) {
    map<int, int> factors;
    primeFactorize(n, factors);

    cout << n << " = ";
    bool first = true;
    for (auto &[prime, count] : factors) {
        if (!first) cout << " × ";
        cout << prime;
        if (count > 1) cout << "^" << count;
        first = false;
    }
    cout << endl;
}

int main() {
    printFactorization(60);  // 60 = 2^2 × 3 × 5
    printFactorization(84);  // 84 = 2^2 × 3 × 7
    return 0;
}
```

应用：计算因子个数

公式：如果 $n = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k}$，则：

- **因子个数**：$d(n) = (a_1 + 1) \times (a_2 + 1) \times \cdots \times (a_k + 1)$
- **因子和**：$\sigma(n) = (1 + p_1 + p_1^2 + \cdots + p_1^{a_1}) \times \cdots \times (1 + p_k + p_k^2 + \cdots + p_k^{a_k})$

```cpp
// 计算因子个数
int countDivisors(int n) {
    int count = 1;
    for (int i = 2; i * i <= n; i++) {
        int exponent = 0;
        while (n % i == 0) {
            exponent++;
            n /= i;
        }
        count *= (exponent + 1);
    }
    if (n > 1) count *= 2;
    return count;
}

// 计算因子和
int sumOfDivisors(int n) {
    int sum = 1;
    for (int i = 2; i * i <= n; i++) {
        int current = 1;
        int term = 1;
        while (n % i == 0) {
            term *= i;
            current += term;
            n /= i;
        }
        sum *= current;
    }
    if (n > 1) sum *= (1 + n);
    return sum;
}
```

数学公式总结

概念	公式	说明
排列数	$P(n,m)=\frac{n!}{(n-m)!}$	考虑顺序的选择
组合数	$C(n,m)=\frac{n!}{m!(n-m)!}$	不考虑顺序的选择
最大公约数	$\gcd(a,b)$	欧几里得算法
最小公倍数	$\operatorname{lcm}(a,b)=\frac{a\times b}{\gcd(a,b)}$	
质因数分解	$n=p_1^{a_1}\times p_2^{a_2}\times\cdots\times p_k^{a_k}$	算术基本定理
因子个数	$d(n)=\prod_{i=1}^{k}(a_i+1)$	

掌握数学基础，提升算法能力，也是高分关键！