进制

进制是一种数字表示系统，用来表示数值的不同方法。它的核心在于用一组固定的符号和规则来表示各种数值。

进制的基本概念

1. **基数（Radix）**

- 基数决定了一个进制系统中所使用的符号数量。例如：
- **二进制（Base-2）**：使用 2 个符号（0 和 1）。
- **十进制（Base-10）**：使用 10 个符号（0 到 9）。
- **十六进制（Base-16）**：使用 16 个符号（0 到 9 和 A 到 F）。
- 一个数的每一位的值由符号与其所在位置的权值共同决定。

2. **位权（Place Value）**

- 每一位上的值与它的基数和位置相关。例如，在十进制中，数字从右到左的位权分别是 $10^0, 10^1, 10^2, \ldots$；在二进制中，位权分别是 $2^0, 2^1, 2^2, \ldots$。

常见的进制系统

1. **十进制（Decimal, Base-10）**

- 我们日常生活中使用的计数系统。
- 符号：0, 1, 2, 3, 4, 5, 6, 7, 8, 9。
- 例子：数字 345 表示 $3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$。

2. **二进制（Binary, Base-2）**

- 计算机系统中使用的进制。
- 符号：0, 1。
- 例子：数字 101 表示 $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$。

3. **八进制（Octal, Base-8）**

- 符号：0, 1, 2, 3, 4, 5, 6, 7。
- 例子：数字 345（八进制）表示 $3 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 = 229$（十进制）。

4. **十六进制（Hexadecimal, Base-16）**

- 符号：0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F（A 表示 10，B 表示 11，以此类推）。
- 例子：数字 1F（十六进制）表示 $1 \times 16^1 + 15 \times 16^0 = 31$（十进制）。

K 进制转十进制

K进制转十进制是将一个以 **k** 为基数的数字表示转换为十进制数的过程，其中 **k** 是一个大于等于2的整数（即二进制、三进制等）。在这个过程中，每一位的权值都是 **k** 的幂，最终将所有位的数值加起来得到十进制数。

转换过程：

假设有一个 **k** 进制数 $d = d_n d_(n-1) \ldots d_1 d_0$，其每个 d_i 是该进制数的一个数字，其中 `0 <= d_i < k`。对应的十进制数 `D` 可以表示为：

$$D = d_n \times k^n + d_(n-1) \times k^{n-1} + \ldots + d_1 \times k^1 + d_0 \times k^0$$

例子 1：二进制转十进制

假设我们有一个二进制数 `1011`（即 `k = 2`），要将其转换为十进制：

$1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

计算过程：

- $1 \times 2^3 = 8$
- $0 \times 2^2 = 0$
- $1 \times 2^1 = 2$
- $1 \times 2^0 = 1$

将结果加起来：
$8 + 0 + 2 + 1 = 11$

所以，`1011`（二进制）等于 **11**（十进制）。

例子 2：十六进制转十进制

假设我们有一个十六进制数 `2F`（即 `k = 16`），要将其转换为十进制：

$2F_{16} = 2 \times 16^1 + 15 \times 16^0$

计算过程：

- $2 \times 16^1 = 32$
- $15 \times 16^0 = 15$（注意，F 表示 15）

将结果加起来：
$32 + 15 = 47$

所以，`2F`（十六进制）等于 **47**（十进制）。

- C++

```cpp
int calc(char c){
    if(c >= 'A'){ // 如果字符是字母（即大于 '9'）
        return 10 + c - 'A'; // 转换为对应的数字值（A=10, B=11, ...）
    }
    return c - '0'; // 如果是数字字符，直接转换为对应的数字值
}

int change(int k, string s){
    int ans = 0;
    for(int i = 0; i < s.size(); i++){
        ans = ans * k + calc(s[i]); // 基于进制 k 进行累加
    }
    return ans; // 返回转换后的十进制数
}

```

- Java

```java
public class Main {
    // 将字符转换为对应的数字值（'0'-'9' 转换为 0-9，'A'-'Z' 转换为 10-35）
    public static int calc(char c) {
        if (c >= 'A') {
            return 10 + c - 'A'; // 如果字符是字母（A=10, B=11, ...）
```

```
        }
        return c - '0'; // 如果是数字字符，直接转换为数字
    }

    // 将给定的字符串 s 转换为十进制数，k 是目标进制
    public static int change(int k, String s) {
        int ans = 0;
        for (int i = 0; i < s.length(); i++) {
            ans = ans * k + calc(s.charAt(i)); // 基于进制 k 进行累加
        }
        return ans; // 返回转换后的十进制数
    }
}

```

- Python

```python
# 将字符转换为对应的数字值（'0'-'9' 转换为 0-9，'A'-'Z' 转换为 10-35）
def calc(c):
    if c >= 'A':
        return 10 + ord(c) - ord('A')  # 如果字符是字母（A=10，B=11，...）
    return ord(c) - ord('0')  # 如果是数字字符，直接转换为数字

# 将给定的字符串 s 转换为十进制数，k 是目标进制
def change(k, s):
    ans = 0
    for char in s:
        ans = ans * k + calc(char)  # 基于进制 k 进行累加
    return ans  # 返回转换后的十进制数

# 示例
k = 16  # 示例进制（可以是其他进制）
s = "1A3"  # 示例字符串
print(change(k, s))  # 打印转换后的十进制数

```

十进制转 K 进制

十进制转 `k` 进制 是将一个十进制数（即我们常用的数字系统，基数为10）转换为任意基数为 `k` 的数字系统（其中 `k` 是大于1的整数）。这种转换通过不断除以 `k`，并将每次的余数记录下来，直到商为零为止。得到的余数的顺序是从最低位到最高位，因此最后的结果需要将余数倒序排列。

过程：

1. **除法和余数**：
   每次将给定的十进制数除以目标进制 `k`，得到商和余数。
   - 商：用于下一次迭代。
   - 余数：即当前位的数字。
2. **记录余数**：
   每次除法后的余数就是该位的数字。从最低位（个位）到最高位（最高位），通过记录余数来构造目标进制的数字。
3. **倒序排列**：
   由于余数是从低位到高位记录的，最后的结果需要倒序排列才能得到正确的 `k` 进制表示。

4. **特殊情况**：
   如果输入的十进制数为 0，那么无论转换为任何进制，结果都是 "0"。

示例：

假设我们要将十进制数 `45` 转换为 `k = 2`（二进制）：

- 第一步：45 除以 2，商为 22，余数为 1（低位）。
- 第二步：22 除以 2，商为 11，余数为 0。
- 第三步：11 除以 2，商为 5，余数为 1。
- 第四步：5 除以 2，商为 2，余数为 1。
- 第五步：2 除以 2，商为 1，余数为 0。
- 第六步：1 除以 2，商为 0，余数为 1（高位）。

得到的余数顺序是 `1, 0, 1, 1, 0, 1`，倒序排列后得到二进制表示为 `101101`，即十进制的 `45` 转换为二进制是 `101101`。

- C++

```cpp
string change(int x, int k) {
    string ans = "";  // 初始化空字符串，用于存储转换结果
    while (x != 0) {
        int t = x % k;  // 计算 x 对 k 取余，得到当前位的值

        // 如果余数小于等于9，直接将其转换为字符并追加到结果字符串
        if (t <= 9) {
            ans = ans + (char)('0' + t);  // '0' + t 转为对应数字字符
        } else {
            // 如果余数大于9，表示需要用字母 A, B, C...表示（例如10为A，11为B等）
            ans = ans + (char)('A' + t - 10);  // 'A' + (t - 10) 转为对应字母字符
        }
        x /= k;  // 更新 x，x 除以 k
    }

    // 将结果字符串反转并返回，返回的字符串为转换后的进制表示
    return reverse(ans.begin(), ans.end());  // reverse 是反转字符串的标准操作
}

```

- Java

```java
public static String change(int x, int k) {
    StringBuilder ans = new StringBuilder(); // 用于构建结果字符串
    while (x != 0) {
        int t = x % k; // 计算当前位的余数
        if (t <= 9) {
            ans.append((char) ('0' + t)); // 数字部分直接添加
        } else {
            ans.append((char) ('A' + t - 10)); // 字母部分从 'A' 开始
        }
        x /= k; // 更新 x，除以进制 k
    }
    return ans.reverse().toString(); // 反转字符串并返回结果
}
```

- Python

```python
    ans = ""  # 初始化结果字符串
    while x != 0:
        t = x % k  # 计算当前位的余数
        if t <= 9:
            ans += str(t)  # 如果余数在 0-9 之间，直接转为字符串
        else:
            ans += chr(ord('A') + t - 10)  # 如果余数大于 9，用字母表示
        x //= k  # 更新 x，除以进制 k

    return ans[::-1]  # 反转结果字符串后返回
```

穿越时空之门

- C++

```cpp
#include<bits/stdc++.h>
using namespace std;

// 计算x在b进制下的各位数字之和
int sumB(int x, int b){
    int sum = 0;
    while(x){
        sum = sum + x % b;  // 取x在b进制下的当前位，并加到sum
        x /= b;  // 将x除以b，继续处理下一位
    }
    return sum;  // 返回各位数字之和
}

// 检查x是否在二进制和四进制下的数字和相等
int check(int x){
    return sumB(x, 2) == sumB(x, 4);  // 比较二进制和四进制下数字和是否相等
}

int main(){
    int ans = 0;  // 用来记录满足条件的数字个数
    for(int i = 1; i <= 2024; i++){  // 遍历1到2024的每个数字
        if(check(i))  // 如果check(i)返回真，说明满足条件
            ans++;  // 满足条件的数字个数加1
    }
    printf("%d", ans);  // 输出满足条件的数字总数
}
```

- Java

```java
public class Main{

    // 计算x在b进制下的各位数字之和
    public static int sumB(int x, int b) {
        int sum = 0;
        while (x > 0) {
            sum += x % b;  // 取x在b进制下的当前位，并加到sum
            x /= b;  // 将x除以b，继续处理下一位
        }
        return sum;  // 返回各位数字之和
```

```java
    }

    // 检查x是否在二进制和四进制下的数字和相等
    public static boolean check(int x) {
        return sumB(x, 2) == sumB(x, 4);  // 比较二进制和四进制下数字和是否相等
    }

    public static void main(String[] args) {
        int ans = 0;  // 用来记录满足条件的数字个数
        for (int i = 1; i <= 2024; i++) {  // 遍历1到2024的每个数字
            if (check(i)) {  // 如果check(i)返回真，说明满足条件
                ans++;  // 满足条件的数字个数加1
            }
        }
        System.out.println(ans);  // 输出满足条件的数字总数
    }
}
```

- Python

```python
# 计算x在b进制下的各位数字之和
def sumB(x, b):
    total_sum = 0
    while x:
        total_sum += x % b  # 取x在b进制下的当前位，并加到total_sum
        x //= b  # 将x除以b，继续处理下一位
    return total_sum  # 返回各位数字之和

# 检查x是否在二进制和四进制下的数字和相等
def check(x):
    return sumB(x, 2) == sumB(x, 4)  # 比较二进制和四进制下数字和是否相等

# 主程序部分
ans = 0  # 用来记录满足条件的数字个数
for i in range(1, 2025):  # 遍历1到2024的每个数字
    if check(i):  # 如果check(i)返回真，说明满足条件
        ans += 1  # 满足条件的数字个数加1

print(ans)  # 输出满足条件的数字总数
```