PAT 甲级刷题实录——1010

作者：aopstudio
原文链接：https://ld246.com/article/1579586075576
来源网站：链滴

原题链接

https://pintia.cn/problem-sets/994805342720868352/problems/994805507225665536

思路

这题是到目前为止比较难的一题，评测系统的通过率也只有 0.11。

首先需要理解基本题意。题目的要求是给一个已知进制的数，求能不能找出一个进制使得另一个未知制的数在该进制下和已知进制的数数值相等。大部分人应该都会想到将两个数的数值都转换为十进制做比较。

在理解了基本题意之后，做的过程中发现这题还有不少坑。

1. 进制是没有上限的。不要认为36即是最大进制，只是35是一位上的最大数字而已。

2. 因为进制没有上限，所以转换为十进制后的数也可能相当相当大，因此用int已经不能满足了，需使用long long类型存储进制和数值。

3. 不能用顺序遍历一个个尝试可能的进制，这样的话会运行超时，需要使用二分法寻找进制。二分法找进制的下界是未知进制数的最大数字+1，因为每一位的数字都不能超过进制数，比如十进制的数字只能是0-9，不可能会有abcd。二分法查找进制的上界是已知进制数的数值，即假设已知进制数为6未知进制数有意义的最小值为10，这时进制刚好是6，如果进制比6还大的话就不可能和6相等，而如未知进制数比10还小那就和进制无关了，因为所有进制个位的单位都是1。

4. 当寻找的可能进制有多个的时候，需要找到最小的那个，这个也是题目中要求的，不过大部分人可在忙于解决其他坑的时候就把这个要求给忘了

代码

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

long long transfer(char c);
long long calculate(string n, long long radix);
long long countMax(string n);
long long binary(string n, long long l, long long r, long long tagValue);
int main()
{
    string n1, n2;
    int tag;
    long long radix, tagValue;    //用十进制表示的数值
    long long l = 2, r, mid;
    cin >> n1 >> n2 >> tag >> radix;
    if (tag == 2)
        swap(n1, n2);
    tagValue = calculate(n1, radix);
    l = countMax(n2)+1;
    r = tagValue;
    l = binary(n2, l, r, tagValue);
    if (calculate(n2, l) == tagValue)    //看看是真正找到了最小进制数还是不可能
```

```
        cout << l;
    else
        cout << "Impossible";
    return 0;
}

long long transfer(char c)    //字符数字转换器
{
    if (c >= '0' && c <= '9')
        return c - '0';
    if (c >= 'a' && c <= 'z')
        return c - 'a' + 10;
    return -1;    //输入其他字符，错误
}

long long calculate(string n, long long radix)    //将特定进制下的数转换为十进制的数值
{
    long long decValue = 0;
    long long exp = 1;
    for (int i = n.length() - 1; i >= 0; i--)
    {
        decValue += transfer(n[i]) * exp;
        exp *= radix;
        if (decValue < 0 || exp < 0)
            return -1;
    }
    return decValue;
}

long long countMax(string n)    //查找字符串中最大的数字,即最低进制-1
{
    long long max = 0;
    for (int i = 0; i < n.length(); i++)
    {
        if (transfer(n[i]) > max)
            max = transfer(n[i]);
    }
    return max;
}

long long binary(string n2, long long l, long long r, long long tagValue)    //二分法比较
{
    long long mid;
    long long value;
    while (l <= r)
    {
        mid = (l + r) / 2;
        value = calculate(n2, mid);
        if (value >= tagValue || value < 0)    //可能进制过大导致value溢出
            r = mid - 1;
        else if (value < tagValue)
            l = mid + 1;
    }
    return l;
```

```
}
```

解释

其他的地方相信大家都能看明白，不过在二分法内部有个需要注意的地方。我把大于和等于写在了一，并且到最后才输出l作为结果。而一般常规的二分法则是等于的时候直接输出结果。这样做的意义在确保l是最小可能的进制数，也就是满足第四个坑的要求，大家可以想一想为什么。当然，返回的l也能是根本没找到后的结果，因此在执行二分法之后还需要加一行语句判断l是否是真正的结果。