PAT 甲级题目讲解：1010《Radix》

题目简介

给定两个正整数 $N1$ 和 $N2$，数字可能包含字母（`0-9`、`a-z`）表示。

现已知其中一个数的进制 $radix$，要求求出另一个数的进制 $radix$，使得两数相等，即 $N1(r1) = N2(r2)$。

若存在多个解，输出最小可能解；若无解，输出 `Impossible`。

样例分析

输入样例 1：

```
6 110 1 10
```

分析：

- 第三个输入 `1` 表示第一个数 $N1$ 的 $radix$ 已知为 `10`（即十进制的 6）；
- $N2 = 110$，若其 $radix = 2$，则对应值为 $1 \times 2^2 + 1 \times 2^1 + 0 = 6$，相等。

输出：

```
2
```

输入样例 2：

```
1 ab 1 1
```

分析：

- $1(2) = ab(r2)$ 求 $r2$，等式左边转十进制为 1；
- ab 在最小合法进制（12）下值已远大于 1，该等式永远不可能成立。

输出：

```
Impossible
```

解题思路

变量说明

变量名	类型	含义
`a, b`	string	两个字符串表示的数字
`t`	int	tag，表示已知 $radix$是第几个数（1 或 2）
`rd`	int	已知数的 $radix$
`f(x, r)`	function	把字符串 x 以进制 r 转成十进制（如失败返回 -1）
`n1`	long long	已知数转换为十进制的结果
`k`	int	未知数中最大数位
`l, r`	long long	二分查找区间

☑ Step 1：预处理统一格式

若 `t == 2`，说明第二个数的 $radix$ 已知，交换 `a` 和 `b`，确保 `a` 是已知进制，统一转换成对 `b` 的进制求解：

```
cin >> a >> b >> t >> rd;
if(t == 2) swap(a, b);
```

☑ Step 2：将已知进制的 `a` 转换为十进制 `n1`

我们需要将字符串 `a` 从已知进制 `rd` 转换为十进制数值 `n1`，以便后续与另一个未知进制的数进行比较。

由于输入可能包含小写字母（即 `a` ~ `z`）与数字（即 `0` ~ `9`），我们要先将每一位字符映射为对应的十进制数值：

- `'0'` ~ `'9'` 对应 $0 \sim 9$；
- `'a'` ~ `'z'` 对应 $10 \sim 35$；

进制转换的数学推导

设字符串 $x = d_0 d_1 d_2 \dots d_{k-1}$，在进制 r 下的十进制表示为：

$$\mathrm{val}(x, r) = d_0 \cdot r^{k-1} + d_1 \cdot r^{k-2} + \cdots + d_{k-1} \cdot r^0$$

在程序实现中，为避免计算幂次，采用左乘右加的迭代方式：

$$\mathrm{val} = (((d_0 \times r + d_1) \times r + d_2) \times r + \cdots + d_{k-1})$$

具体实现过程：

1. 使用 `long long s` 保存累加结果；
2. 每次将当前结果 `s` 左移一位（即 `s * r`），再加上当前数位值 `t`；
3. 为防止溢出，判断乘加前是否可能超过 `LLONG_MAX`，防止不合法解误判为合法。

```
    long long f(string x, int r){
        long long s = 0;
        for(int i = 0; i < x.size(); i++){
            int t = (x[i] <= '9') ? (x[i] - '0') : (x[i] - 'a' + 10);
            if(s > (LLONG_MAX - t) / r) return -1; // 溢出保护
            s = s * r + t; // 将当前结果左移一位，再加上当前数位值 t
        }
        return s;
    }

    long long n1 = f(a, rd);
```

☑ Step 3：确定可能的 radix 区间

本题本质是**通过二分枚举未知进制** $radix$，使得 `f(b, radix) == n1`。为了让二分有效进行，我们需要先确定可能的值 $radix$ 域范围，即：**最小可能** $radix$ **和 最大可能** $radix$。

最小 radix（下界）

一个数字在某个进制下合法，要求其中每一位的数值 $d_i < r$，即：

$$r > \max\{d_0, d_1, \ldots, d_{k-1}\}$$

所以对于字符串 `b`，我们需找出其所有字符中最大的数位值 `k`，那么：

$$radix_{min} = k + 1$$

否则该进制下 `b` 中可能出现非法位。

最大 radix（上界）

我们需要寻找的 $radix$ 满足：

$$f(b, \text{radix}) = f(a, rd) = n_1$$

设 n_1 是已知数 `a` 在其 $radix$ `rd` 下转换为十进制的值。考虑：

- $f(b, radix)$ 随 $radix$ 增大而单调递增；
- 若 $radix$ 太大，则 `f(b, radix) > n1`；
- 所以最大 $radix$ 的边界为 n_1。

换句话说：若 $radix > n_1$，最小可能得到的值也会超过 n_1，肯定不等于 n_1，无需再查。

实现代码如下：

要使 `f(b, radix) == n1`，则：

- `b` 中最大数位记为 `k`，最小可能 radix 为 `k + 1`；
- 最大 radix 不会超过 `n1`：

```
int k = 0;
for(char ch : b){
    int val = (ch <= '9') ? ch - '0' : ch - 'a' + 10;
    k = max(k, val);
}
long long l = k + 1, r = max(l, n1); // 最小可能 radix，最大不超过 n1（确保区间合法）
```

☑ Step 4：二分查找使得 `f(b, radix) == n1` 的最小 $radix$

我们已获得 `radix ∈ [k+1, n1]` 的可能区间。

现在的任务是：**在该区间内查找最小的 $radix$，使得** `f(b, radix) == n1`。

这是一个典型的"**单调性 + 最值**"问题，适合使用**二分查找**解决。

观察函数 `f(b, radix)` 的性质：

- $radix$ 越大，`f(b, radix)` 的结果也越大（单调递增）；
- 一旦 $radix$ 太大，`f()` 会超出 long long 范围或超过 n_1；
- 所以我们可以对 $radix$ 进行单调性剪枝，快速缩小范围；

🎯 目标转化为"最小满足条件的 radix"

设：

- $n_1 = f(a, rd)$ 为已知十进制目标值；
- 当前枚举 $radix$ 为 mid，计算 $n_2 = f(b, mid)$：

判断分三种情况：

1. 若 $n_2 < n_1$：说明 $radix$ 偏小，应往右侧逼近：
$$l = mid + 1$$
2. 若 $n_2 > n_1$ 或 $n_2 = -1$（溢出）：
$$r = mid - 1$$
3. 若 $n_2 = n_1$：
    - 说明找到一个合法 $radix$；
    - 但要继续往左搜索更小合法解，直到收敛：
$$r = mid - 1$$

最终，l 会停在最小满足条件的 $radix$ 处。

🔁 实现代码如下：

```
while(l <= r){
    long long mid = (l + r) >> 1;
    long long n2 = f(b, mid);
    if(n2 == -1 || n2 > n1) r = mid - 1;
    else if(n2 < n1) l = mid + 1;
    else r = mid - 1; // 继续向左逼近最小解
}
```

☑ Step 5：判断是否找到解

根据循环退出条件 `l > r`，可知：

- 若存在解，则 `l` 就是最小合法 $radix$；
- 若不存在解，则 `f(b, l) != n1`，输出 `Impossible`

```
if(f(b, l) == n1) cout << l;
else cout << "Impossible";
```

☑ 完整代码（C++）

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

string a, b;
int t, rd;

long long f(string x, int r){
    long long s = 0;
    for(int i = 0; i < x.size(); i++){
        int t = (x[i] <= '9') ? (x[i] - '0') : (x[i] - 'a' + 10);
        if(s > (LLONG_MAX - t) / r) return -1;
        s = s * r + t;
    }
    return s;
}

int main(){
    cin >> a >> b >> t >> rd;
    if(t == 2) swap(a, b);
    long long n1 = f(a, rd);
    int k = 0;
    for(int i = 0; i < b.size(); i++){
        int t = (b[i] <= '9') ? (b[i] - '0') : (b[i] - 'a' + 10);
        k = max(k, t);
    }
    long long l = k + 1, r = max(l, n1);
    while(l <= r){
        long long mid = (l + r) >> 1;
        long long n2 = f(b, mid);
        if(n2 == -1 || n2 > n1) r = mid - 1;
        else if(n2 < n1) l = mid + 1;
        else r = mid - 1;
    }
    if(f(b, l) == n1) cout << l;
    else cout << "Impossible";
    return 0;
}
```

常见错误提醒

错误类型	具体表现
未处理 `tag == 2`	没有 swap 导致已知 $radix$ 错位
转换函数未防止溢出	进制乘法可能导致 long long 溢出
$radix$ 范围确定错误	忽略了 `k+1` 的限制，或者最大值不合理

☑ 总结归纳

📌 核心方法总结

- 字符串进制转换函数的设计；
- $radix$ 的合法范围确定：$[k+1, n1]$；
- 二分查找最小合法 $radix$ 满足条件。

📋 技术要点回顾

- 字符转十进制技巧：`'0'` ~ `'9'` 对应 $0 \sim 9$, `'a'` ~ `'z'` 对应 $10 \sim 35$；；
- 二分查找策略（最小满足条件）；
- LLONG_MAX 防止溢出技巧。

📊 复杂度分析

- 时间复杂度：$\mathcal{O}(L \log N)$，L 为字符串长度，N 为最大可能 $radix$
- 空间复杂度：$\mathcal{O}(1)$

🧠 思维拓展

- 如果输出最大合法 $radix$ 解，该如何修改？
- 若 $radix$ 不限定在 36 内（允许更大进制），如何支持？