PAT_A1010. Radix(25)

1.Abstraction

1.1 Algorithm and idea

```
1.进制转换。
    string snum;snum中储存了初始的数据
    int convert(char c);将字字符转化为整形
    int sum=0;
    for(int i=0;i<snum.len();++i){
        sum=sum*radix+convert(snum[i]);
    }
  这是一个很精巧的进制算法，免去了将snum转置的步骤
2.二分查找。
    由于radix是严格递增的，所以可以使用二分查找。如果暴力求解会超时。
```

1.2 Notice

```
1.越界问题。题目没有完全明确给出一些问题。比如，原始数据不可能越界，但是经过进制转换之后就会发生越界。
    比如
        a b 1 1不会超过long long
        b在运算过程中可能发生越界的问题
2.给定一个数据a,如果a的某一为最大为x，那么实际上a的进制，最小不能小于x+1。要注意处理。
```

2.Problem:Radix

Given a pair of positive integers, for example, 6 and 110, can this equation 6 = 110 be true? The answer is "yes", if 6 is a decimal number and 110 is a binary number.

Now for any pair of positive integers N1 and N2, your task is to find the radix of one number while that of the other is given.

Input Specification:

Each input file contains one test case. Each case occupies a line which contains 4 positive integers: N1 N2 tag radix Here N1 and N2 each has no more than 10 digits. A digit is less than its radix and is chosen from the set {0-9, a-z} where 0-9 represent the decimal numbers 0-9, and a-z represent the decimal numbers 10-35. The last number "radix" is the radix of N1 if "tag" is 1, or of N2 if "tag" is 2.

Output Specification:

For each test case, print in one line the radix of the other number so that the equation N1 = N2 is true. If the equation is impossible, print "Impossible". If the solution is not unique, output the smallest possible radix.

Sample Input 1:

```
6 110 1 10
```

Sample Output 1:

```
2
```

Sample Input 2:

```
1 ab 1 2
```

Sample Output 2:

```
Impossible
```

3.Algorithm note

3.1进制转换

```
    string snum;//snum中储存了初始的数据
    int convert(char c);//将字符转化为整形
    int sum=0;
    for(int i=0;i<snum.len();++i){
        sum=sum*radix+convert(snum[i]);
    }
```

3.2二分查找算法

二分查找算法主要有两种模板。两种模板的二分区间都为闭区间

一种是*精确查找*符合某个条件数据，如果查找不到要指明不存在。

另一种是*模糊查找*，要求查找到第一个符合某种条件的数据，如果查找不到，返回该数据应该出现的位置。

两种模板应该理解并且熟练掌握。

3.2.1要点

1.左右界的剔除与返回值的关系

2.while循环的条件

3.二分区间全部为闭区间

4.后面给出的low_bound和upper_bound在标准库中都有

3.2.2精确查找

```
//A[]为严格递增序列，left为二分下界，right为二分上界，x为欲查找的数值
//二分区间闭区间[left,right]
int binarySearch(int A[],int left,int right,int x){
    int mid;
    //注意这里的while循环，并且区分与模糊查找的不同
    while(left<=right){
        mid=(left+right)/2;
        if(x<A[mid]){
            //右界将不符合条件的全部剔除
            right=mid-1;
        }
        else if(A[mid]<x){
            //左界将不符合条件的全部剔除
            left=mid+1;
        }
        else if(A[mid]==x){
            return mid;
        }
    }
    return -1;
}
```

3.2.3模糊查找

```
int solve(int left,int right){
    int mid;
    //注意这里的while循环的条件
    while(left<right){
        mid=(left+right)/2;
        if(条件成立){
            //注意right没有剔除作用
            right=mid;
        }
        else{
            //注意left将所有条件不成立的都剔除出去了
            left=mid+1;
        }
    }
    //返回的是left
    return left;
}
```

```
//A[]为递增序列，x为想要查询的数，函数返回第一个大于等于x的元素的位置
int low_bound(int A[],int left,int right,int x){
    int mid;
    while(left<right){
        mid=(left+right)/2;
        if(A[mid]>=x){
            right=mid;
        }
        else{
            left=mid+1;
        }
    }
    return left;
}
```

```
//A[]为递增序列，x为想要查询的数，函数返回第一个大于x的元素的位置
int upper_bound(int A[],int left,int right,int x){
    int mid;
    while(left<right){
        mid=(left+right)/2;
        if(A[mid]>x){
            right=mid;
        }
        else{
            left=mid+1;
        }
    }
    return left;
}
```

4. Code

4.1 Edit 0:

4.1.1 Algorithm abstraction

```
```

4.1.2 Notice

```
```

4.1.3 Code Block

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using std::max;
using std::min;

typedef long long LL;

LL Map[256];


LL inf=(1LL<<63)-1;

void init(){
    for(char c='0';c<='9';c++){
        Map[c]=c-'0';
    }
    for(char c='a';c<='z';c++){
        Map[c]=c-'a'+10;
    }
}

LL convertNum10(char number[],LL radix,LL t){
    LL ans=0;
    int len=strlen(number);
    for(int i=0;i<len;++i){
        ans=ans*radix+Map[number[i]];
        if(ans<0||ans>t) return -1;
    }
    return ans;
}

int cmp(char N2[],LL radix,LL t){
    int len=strlen(N2);
    LL num=convertNum10(N2,radix,t);
    if(num<0) return 1;
    if(t>num) return -1;
    else if(t==num) return 0;
    else return -1;
}

LL binarySearch(char N2[],LL left,LL right,LL t){
    LL mid;
    while(left<=right){
        mid=(left+right)/2;
        int flag=cmp(N2,mid,t);
        if(flag==0) return mid;
        else if(flag==-1) left=mid+1;
        else right=mid-1;
    }
    return -1;
}


int findLargestDigit(char N2[]){
    int ans=-1,len=strlen(N2);
    for(int i=0;i<len;++i){
        if(Map[N2[i]]>ans){
            ans=Map[N2[i]];
        }
    }
    return ans+1;
}

char N1[20],N2[20],temp[20];

int tag,radix;

int main(){
    init();
    scanf("%s %s %d %d",N1,N2,&tag,&radix);
    if(tag==2){
        strcpy(temp,N1);
        strcpy(N1,N2);
        strcpy(N2,temp);
    }

    LL t=convertNum10(N1,radix,inf);
    LL low=findLargestDigit(N2);
    LL high=max(low,t)+1;
    LL ans=binarySearch(N2,low,high,t);
    if(ans==-1) printf("Impossible\n");
    else printf("%lld\n",ans);
    return 0;
}
```

4.2 Edit 1:

4.2.1 Algorithm abstraction

```
```

4.2.2 Notice

```
```

4.2.3 Code Block

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using std::max;
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;


typedef long long LL;
const LL INF=(1LL<<63)-1;
LL MAP[256];

void init_MAP(){
    for(char c='0';c<='9';++c){
        MAP[c]=c-'0';
    }
    for(char c='a';c<='z';++c){
        MAP[c]=c-'a'+10;
    }
    return;
}

LL convertNum10(string snum,LL radix){
    LL ans=0;
    int len=snum.size();
    for(int i=0;i<len;++i){
        ans=ans*radix+MAP[snum[i]];
    }
    return ans;
}

//题目没有明确给出，但是给定进制的数不过超过long long 的范围
//一旦经过转换溢出即说明该数较大

LL cmp(LL t,string snum,LL radix){
    LL num=convertNum10(snum,radix);
    if(num<0) return -1;
    else if(t>num) return 1;
    else if(t<num) return -1;
    else if(t==num) return 0;
    else{
        return 10086;
    }
}

LL binarySearch(LL t,string snum,LL left,LL right){
    LL mid;
    while(left<=right){
        mid=(left+right)/2;
        int flag=cmp(t,snum,mid);
        if(flag==0){
            return mid;
        }
        else if(flag<0){
            right=mid-1;
        }
        else if(flag>0){
            left=mid+1;
        }
    }
    return -1;
}

LL findLargest(string snum){
    LL max=0;
    int len=snum.size();
    for(int i=0;i<len;++i){
        if(MAP[snum[i]]>max){
            max=MAP[snum[i]];
        }
        else{
            ;
        }
    }
    return max+1;
}


int main(){
    init_MAP();
    string sn1,sn2;
    LL flag;
    LL radix;
    cin>>sn1>>sn2>>flag>>radix;
    LL n;
    string sn;
    if(flag==1){

        n=convertNum10(sn1,radix);
        sn=sn2;
    }
    else if(flag==2){
        n=convertNum10(sn2,radix);
        sn=sn1;
    }
    else{
        ;
    }


    LL low=findLargest(sn);
    LL high=max(low,n)+1;
    LL ans=binarySearch(n,sn,low,high);

    if(ans==-1){
        cout<<"Impossible"<<endl;
    }
    else{
        cout<<ans<<endl;
    }
    return 0;
}
```

5. Summary

> 注意掌握二分查找的各种变形