题解1：

首先，如果A与B的位数是不相同的，那么我们只需要判断它们的位数多少来判断两个数的大小，直接输出答案就可以了。在它们位数相同并且A>=B的情况下，我们可以采用贪心的策略，依次与A的每一位去进行比较，当必须移动数字时，我们再进行移动，更新答案。当然，直接的模拟移动是不可取的，我们可以用链表来模拟，或者记录一下B的每个数字在B内的位置，移动之后标记一下即可，每次移动优先选择原位置靠后的来前移，这样可以保证B增大的值最大。在进行比较的过程中有一点要注意的就是若当前比较的两位数字相等，并且B的后面有大于A当前位的数字，那么要把目前移动次数+1去更新一下答案，然后按不移动继续比较。

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <cmath>

#include <algorithm>

#include <queue>

#include <map>

#include <set>

#include <vector>

#include <string>

#include <stack>

#define INF 0x3f3f3f3f

#define eps 1e-8

using namespace std;

typedef long long LL;

const int maxn=1e5+5;

char A[maxn],B[maxn];

int cnt[10];

bool use[maxn];

vector <int> pos[10];

int now[10];

int main()

{

int T;

scanf("%d",&T);

while(T--)

{

scanf("%s%s",A,B);

int lena=strlen(A),lenb=strlen(B);

if(lena!=lenb)

{

puts(lena<lenb?"0":"Poor LoveZx");

continue;

}

for(int i=0;i<10;++i) pos[i].clear();

memset(cnt,0,sizeof(cnt));

for(int i=0;i<lenb;++i)

{

pos[B[i]-'0'].push\_back(i);

cnt[B[i]-'0']++;

}

memset(use,0,sizeof(use));

int tmp=0,ans=INF,cur=0,maxd;

for(int i=0;i<10;++i) now[i]=pos[i].size()-1;

for(int i=9;i>=0;--i) if(pos[i].size()) {maxd=i;break;}

bool poor=1;

for(int i=0;i<lena;++i)

{

while(use[cur]) cur++;

while(!cnt[maxd]) maxd--;

if(A[i]>B[cur])

{

if(maxd>A[i]-'0')

{

ans=min(ans,tmp+1);

poor=0;

break;

}

else if(maxd<A[i]-'0') break;

tmp++;

use[pos[maxd][now[maxd]]]=1;

now[maxd]--;

cnt[maxd]--;

}

else if(A[i]==B[cur])

{

if(maxd>A[i]-'0')

{

ans=min(ans,tmp+1);

poor=0;

}

cnt[B[cur]-'0']--;

cur++;

}

else

{

ans=min(ans,tmp);

poor=0;

break;

}

}

if(poor) puts("Poor LoveZx");

else printf("%d\n",ans);

}

return 0;

}

题解2：

首先，如果A与B的位数是不相同的，那么我们只需要判断它们的位数多少来判断两个数的大小，直接输出答案就可以了。在两数位数相同时，我们还要考虑B能否大于A——将B的每一位按从大到小排序，再判断排完的数是否大于A即可（注意A、B都是字符串）。

在B能够大于A的情况下，我们需要求最小值。此时我们需要采取贪心的策略：从最高位开始，将B的每一位与A的每一位比较，如果小于A就从后面移动数字上来填充（优先移动低位）。

但是由于这题实现要求比较严格，所以我们必须O(n)地实现（即不能模拟移动数字的过程）。这时我们可以设一个变量cnt指向当前B中与A[i]比较的那一位，同时统计出B[cnt]之后还有每一位数（0，1，2，3，……，9）还各有几个counts[0], counts[1],…. counts[9],。如果遇到需要移动的情况，就把需要移动的数的数目减去即可。这样就可以避免模拟的情况出现。

代码

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <functional>

int counts[20];

char sa[100010], sb[100010];

char temps[100010];

int dp[100010];

int main()

{

// freopen("input.txt","r",stdin);

// freopen("output.txt","w",stdout);

int CAS;

scanf("%d", &CAS);

while (CAS--)

{

scanf("%s%s", sa, sb);

int la=strlen(sa), lb=strlen(sb);

if (lb>la)

printf("0\n");

else if (lb<la)

printf("Poor LoveZx\n");

else

{

memcpy(temps, sb, sizeof(sb));

std::sort(temps, temps+lb, std::greater<char>() );

if (strcmp(temps, sa)!=1)

printf("Poor LoveZx\n");

else

{

memset(counts, 0, sizeof(counts));

for (int i=1; i<=lb; i++)

counts[sb[i-1]-'0']++;

int resu=lb;

dp[0]=0;

for (int i=1; i<=lb; i++)

{

int cnt=i-dp[i-1];

if (sa[i-1]<sb[cnt-1])

resu=std::min(resu, dp[i-1]);

for (int j=sa[i-1]-'0'+1; j<10; j++)

if (counts[j])

resu=std::min(resu, dp[i-1]+1);

if (sa[i-1]==sb[cnt-1])

dp[i]=dp[i-1];

else

{

if (counts[sa[i-1]-'0']!=0)

{

dp[i]=dp[i-1]+1;

counts[sa[i-1]-'0']--;

}

else

break;

}

}

printf("%d\n", resu);

}

}

}

fclose(stdin);

fclose(stdout);

return 0;

}