## 虫食算 解题报告

## <问题描述>

所谓虫食算,就是原先的算式中有一部分被虫子啃掉了,需要我们根据剩下的数字来 判定被啃掉的字母。来看一个简单的例子:

43#9865#045

+ 8468#6633

其中#号代表被虫子啃掉的数字。根据算式,我们很容易判断:第一行的两个数字分别是5和3,第二行的数字是5。

现在,我们对问题做两个限制:

首先,我们只考虑加法的虫食算。这里的加法是 N 进制加法,算式中三个数都有 N 位,允许有前导的 0。

其次,虫子把所有的数都啃光了,我们只知道哪些数字是相同的,我们将相同的数字用相同的字母表示,不同的数字用不同的字母表示。如果这个算式是 N 进制的,我们就取英文字母表午的前 N 个大写字母来表示这个算式中的 0 到 N-1 这 N 个不同的数字:但是这 N 个字母并不一定顺序地代表 0 到 N-1)。输入数据保证 N 个字母分别至少出现一次。

BADC

+ CRDA

**DCCC** 

上面的算式是一个 4 进制的算式。很显然,我们只要让 ABCD 分别代表 0123,便可以让这个式子成立了。你的任务是,对于给定的 N 进制加法算式,求出 N 个不同的字母分别代表的数字,使得该加法算式成立。输入数据保证有且仅有一组解,

#### - 输入文件

输入文件 alpha.in 包含 4 行。第一行有一个正整数 N(N<=26),后面的 3 行每行有一个由大写字母组成的字符串,分别代表两个加数以及和。这 3 个字符串左右两端都没有空格,从高位到低位,并且恰好有 N 位。

# - 输出文件

输出文件 a l pha. out 包含一行。在这一行中,应当包含唯一的那组解。解是这样表示的:输出 N 个数字,分别表示 A,B,C……所代表的数字,相邻的两个数字用一个空格隔开,不能有多余的空格。

- 样例输入

5

**ABCED** 

**BDACE** 

**EBBAA** 

- 样例输出
- 1 0 3 4 2
- 数据规模

对于 30%的数据,保证有 N<=10;

对于 50%的数据,保证有 N<=15;

对于全部的数据,保证有 N<=26。

# <算法分析>

经典的搜索题。最单纯的搜索的时间复杂度为 0(n!),是会非常严重的超时的。计算机是很"笨"的,它不会思考,在盲目搜索的过程中,很容易出现这种情况:

计算机在某一位搜索出了一个算式1 + 1 = 3,并且继续搜索。

明显,人眼很容易就看出这是不合法的,但计算机不会。于是,我们想到了第一个剪 枝:每次搜索的时候,从最后向前判断是否有不合法的式子。

这一个剪枝非常简单,但是效果却非常的好。因为它剪去了很多不必要的搜索。为了配合这一种剪枝更好的实行,搜索顺序的改变也成为大大提高程序效率的关键:从右往左按照字母出现顺序搜索,有很大程度上提高了先剪掉废枝的情况,使程序的效率得到大大的提高。

有了以上两个剪枝,程序就已经可以通过大部分测试点了。但是有没有更多的剪枝呢? 答案是肯定的。

根据前面的剪枝,我们可以找到类似的几个剪枝:

```
对于 a + b = c 的形式,假如:
A***?***
+ B*?**?**
C***???*
```

其中\*代表已知,?代表未知。那么,A + B与C的情况并不能直接确定。但是,假如 (A + B) % N与(A + B + 1) % N都不等于C的话,那么这个等式一定是不合法的。 因为它只有进位和不进位的两种情况。

同样,我们在一个数组里记录了 Used [i]表示一个数字有没有用过,那么,对于某一位 A + B = C 的等式,如果已经得到了两个数,另一个数还待搜索的时候,我们还可以根据这个加入一个剪枝:

```
例如 A + ? = C的形式,
考虑不进位的情况,则?处为 P1 = (C - A + N) % N
假如考虑进位的情况,则?处为 P2 = (C - A - 1 + N) % N
假如 P1、P2 均被使用过,那么这个搜索一定是无效的,可以剪去。
```

有了以上的剪枝,就可以很轻松地通过所有的测试数据了。当然,还有很多值得思考的剪枝以及其他的思路,例如枚举进位、解方程(但是可能需要枚举)等,在这里就不详细讨论了。

## <代码清单>

```
#include <fstream>
#include <string>
using namespace std;

ifstream fin("alpha.in");
ofstream fout("alpha.out");

bool finish, hash[256], used[27];
int n, stk[27];
string a, b, c;
string word;

void init() {
   fin >> n >> a >> b >> c;
```

```
finish = false;
}
void outsol() {
   int i, ans[27];
   for (i = 0; i < n; i ++)
       ans[word[i] - 65] = stk[i];
   fout << ans[0];
   for (i = 1; i < n; i ++)
       fout << " " << ans[i];
   fout << endl;</pre>
   finish = true;
}
void addup(char ch) {
   if (!hash[ch]) {
       hash[ch] = true;
      word = word + ch;
}
string change(string str, char x, char y) {
   for (int i = 0; i < n; i ++)
       if (str[i] == x)
          str[i] = y;
   return str;
}
void pre_doing() {
   word = "";
   memset(hash, 0, sizeof(hash));
   for (int i = n - 1; i \ge 0; i --) {
       addup(a[i]); addup(b[i]); addup(c[i]);
   }
   memset(used, 0, sizeof(used));
}
bool bad() {
   int p, g = 0;
   for (int i = n - 1; i \ge 0; i --) {
```

```
if (a[i] \ge n \mid\mid b[i] \ge n \mid\mid c[i] \ge n) return false;
       p = a[i] + b[i] + g;
       if (p % n != c[i]) return true;
       g = p / n;
       p %= n;
   }
   return false;
}
bool modcheck() {
   int i, p, p1, p2, g = 0;
   //a + b = c, all know
   for (i = n - 1; i \ge 0; i --) {
       if (a[i] >= n \mid | b[i] >= n \mid | c[i] >= n) continue;
       p = (a[i] + b[i]) \% n;
       if (!(p == c[i] || (p + 1) % n == c[i])) return true;
   }
   //a + ? = c
   for (i = n - 1; i \ge 0; i --) {
       if (!(a[i] < n \&\& c[i] < n \&\& b[i] >= n)) continue;
       p1 = (c[i] - a[i] + n) \% n;
       p2 = (p1 - 1) \% n;
       if (used[p1] && used[p2]) return true;
   }
   //? + b = c
   for (i = n - 1; i \ge 0; i --) {
       if (!(a[i] >= n \&\& c[i] < n \&\& b[i] < n)) continue;
       p1 = (c[i] - b[i] + n) \% n;
       p2 = (p1 - 1) \% n;
       if (used[p1] && used[p2]) return true;
   }
   //a + b = ?
   for (i = n - 1; i \ge 0; i --) {
       if (!(a[i] < n \&\& b[i] < n \&\& c[i] >= n)) continue;
       p1 = (a[i] + b[i]) \% n;
       p2 = (p1 + 1) \% n;
       if (used[p1] && used[p2]) return true;
   }
   return false;
}
```

```
void dfs(int l) {
   int i;
   string A, B, C;
   if (finish) return;
   if (bad()) return;
   if (modcheck()) return;
   if (l == n) {
      outsol();
       return;
   }
   for (i = n - 1; i \ge 0; i --)
       if (!used[i]) {
          used[i] = true; A = a; B = b; C = c;
          a = change(A, word[l], i);
          b = change(B, word[l], i);
          c = change(C, word[l], i);
          stk[l] = i;
          dfs(l + 1);
          used[i] = false; a = A; b = B; c = C;
       }
}
int main() {
   init();
   pre_doing();
   dfs(0);
   return 0;
}
```

# <小结>

搜索题的框架往往不难找到,关键就是在搜索的优化上,本文的主要篇幅也就是讨论了几种有效的优化。搜索问题的优化更多的需要选手的经验和思考、分析问题的能力,所以搜索剪枝也是竞赛中经久不衰的经典问题。