[Problem C]　覆面計算

- 与えられた覆面計算に対して、「等式を満たすような数字の割り当てが何通りあるか」を求める問題であるので、各文字に対する数字の割り当てを順番に生成していき、その割り当てが等式を満たすかどうかチェックすればよい。異なる文字には異なる数値（0～9）が割り当てられるので、最大で 10! ≒ 約 360 万通りの組み合わせをチェックする必要がある。（1 桁の場合を除いて最上位の桁は 0 で無いのでチェックすべき組み合わせの数はもう少し少なくなる。）⇒　あまり工夫しなくても何とかなりそう！！？
- 例えば、「ACM + IBM = ICPC」は
  「100×A + 10×C + M + 100×I + 10×B + M = 1000×I + 100×C + 10×P + C」を意味する。これを整理すると
  「100×A － 91×C + 2×M －900×I + 10×B －10×P = 0」となる。
  この係数を事前に求めておき、A,C,M,I,B,P の全てに数字を割り当てた時にこの式に代入してチェックする。
- 各文字に対する全ての可能な割当を組織的に生成し、等式が成立する回数を数える。全ての可能な割当を組織的に生成する方法（順列の生成）については、以下のプログラム例の再帰的関数 perm を参照。
- 以下のプログラム例を上記の例に適用した場合、main 関数から perm(0)を呼び出す直前には
  nc = 6 (6 種類の文字が現れている)
  c = {'A', 'C', 'M', 'I', 'B', 'P',　・・・}
  割り当て可能な数値のリスト　nu = {0, 1, 2, 3, 4, 5, 6, 7, 8. 9}
  値 0 が割当不可であることを示すリスト　zskip = {1, 0,0,1,0,0,・・・}
  （A, I は 2 桁以上の数値の最上位桁になるので、値 0 を割り当てることはできない）
  等式が成立する割り当ての個数　count = 0
  perm(0)が呼び出されると、その内部では次のように動作する。
  呼び出された時点で、nu[0]～nu[9]に未割当の数値が格納されている。
  for 文の最初の繰り返し(i=0)では、値 0 (= val = nu[0])が割り当てる数値として選ばれるが、この文字に値 0 は割り当てられない(zl = 1) ので何もせずに次の繰り返しに入る。
  for 文の 2 回目の繰り返し（i=1）では、値 1 (= val = nu[1]) が選ばれその値を最初の文字(A)の値として割り当てる。そして、nu[1] = nu[9]により、nu = {0, 9, 2, 3, 4, 5, 6, 7, 8, 9}としてから perm(1)を呼び出している。perm(1)では nu[0]～nu[8]、即ち、{0, 9, 2, 3, 4, 5, 6, 7, 8}が呼び出し時点での未割当数値となっている。
  perm(0)では、perm(1)から戻ってきた時点で nu[1] = val (= 1)により値 1 を未割当数値に戻して for 文の次の繰り返し行っている。
- 以下のプログラム例を Core i7 1.8GHz 上で 4 種類の判定データに適用すると、各々2～3 秒程度で結果が得られる。

プログラム例

```
/* ACM-ICPC2009 国内予選 Problem C */
// http://www.waseda.jp/assoc-icpc2009/preliminary/contest/all_ja.html#section_C
// filename = pc1.c
// コンパイル： cc -O2 pc1.c
// 実行方法：   ./a.out < C0 > C0.result 等
// 確認方法：   diff C0.ans C0.result 等
// アルゴリズム：覆面算に現れるアルファベットに対して互いに異なる数値を割当、
//               その割当が覆面算の等式を満たすかどうか判定する

#include <stdio.h>
#include <string.h>

#define MAXN 12         // 文字列の個数 N の最大値
#define MAXLEN 8        // 文字列の長さの最大値

int n;                  // N, 3 <= n <= 12
char strs[MAXN][MAXLEN+1];      // STRING_i を strs[i-1][ ]に格納
int len[MAXN];          // 各 STRING の長さ
char c[10];             // STRING に表れる文字. 最大でも 10 個
int v[10];              // 各文字に割当てた値
int a[10];              // 各文字の係数
int zskip[10];          // 1 -> 値 0 は割当不可
int nc;                 // 使われている文字数
int weight[] = {1,10,100,1000,10000,100000,1000000,10000000}; // 桁位置の重み
int count;              // 等式を満たす割当の個数
int nu[10];             // 未使用の数値リスト

// 全ての文字に対して数値が割当済みの時は等式が成立するかをチェック
// 数値未割当の文字が残っている場合は、未使用の数値を順番に割当てる
void perm(int level)    // level = 数値割当済みの文字数
{
  int i, val, zl;

  if(level == nc){      // 全ての文字に対して数値割当済みなら
    val = 0;            // その割当に対する条件式の値を計算
    for(i=0; i<nc; i++)
      val += v[i]*a[i];
    if(val == 0) count++;       // 値が 0 なら等式成立
    return;
  }
  zl = zskip[level];            // zl = 1 なら 0 は割当不可
  // 現時点で未使用の数値は nu[0] ～ nu[9-level] に格納されている
  for(i=0; i<10-level; i++){    // 未使用の数値を順に割当てていく
    val = nu[i];                // val は未使用の数値
    if(val==0 && zl) continue;  // 0 が割当不可なら次の未使用数値を割当てる
    v[level] = val;             // val を割当てる
    nu[i] = nu[9-level];        // val は使用済みとする
    perm(level+1);              // 再帰的に次の文字に数値を割当てていく
    nu[i] = val;                // 次の割当を試す前に val を未使用に戻す
  }
}

int main()
```

```
{
  int i,j,k;

  while(1){
    scanf("%d", &n);    // N を入力
    if(n==0) break;     // N=0 なら終了
    for(i=0; i<n; i++){                 // n 個の文字列を読み込む
      scanf("%s", &strs[i][0]);         // 文字列を読み込み
      len[i] = strlen(&strs[i][0]);     // その長さをセット
    }
    nc = 0;
    for(i=0; i<10; i++) a[i] = zskip[i] = 0;
    for(i=0; i<n; i++){
      char *s;
      s = &strs[i][0];                  // s は i 番目の文字列
      for(j=0; j<len[i];j++){
        char d;
        int w;
        d = s[j];                       // d は s の j 番目の文字
        for(k=0; k<nc; k++){            // d が登録済みかどうかをチェック
          if(d == c[k]) break;          // d は登録済み
        }
        if(k==nc) c[nc++] = d;  // k==nc のとき d は新しい文字なので新規登録
        // k は現在注目している文字 (d) が登録されている位置
        if(j==0 && len[i]>1){   // 長さ 2 以上の文字列の最上位桁なら
          zskip[k] = 1;         // 値 0 は割当不可
        }
        w = weight[len[i]-j-1]; // 文字 d が表れる桁位置からその重みを求める
        if(i==n-1) w *= -1;     // 右辺に表れる文字の係数は負
        a[k] += w;
      }
    }
    count = 0;
    for(i=0; i<10; i++) nu[i] = i;      // 未割当数値のリストを初期化（全てみ割当）
    perm(0);    // 全ての可能な割当を組織的に生成して条件式が成立する回数を数える
    printf("%d¥n",count);       // 求まった結果を出力
  }
}
```

- より効率を向上させるには、下位の桁に現れる文字から数値を割当て行き、下位 i 桁の現れる文字に数値が割り当てられた時点で、下位 i 桁の等式が成立するかどうかをチェックし、不成立なら未割当の文字に対する数値割り当てをスキップする方法が考えられる。例えば「ACM + IBM = ICPC」で、M, C, B, P, A, I の順に数値割り当てを行うものとする。この時、M と C に数値を割り当てた時点で下位 1 桁に現れる全ての文字に数値が割り当てられているので、下位 1 桁に関する等式

  $(2\times M - C) \% 10 = 0$

  をチェックし、これが不成立なら残りの文字に対する数値割り当てをスキップする（枝狩）。成立する場合は、残りの文字に対する数値割当を継続する。M, C, B, P まで数値

割当を行うと下位 2 桁に現れる全ての文字に数値が割り当てられているので、下位 2 桁に関する等式

$(9\times C + 2\times M + 10\times B - 10\times P) \% 100 = 0$

をチェックし、これが不成立なら残りの文字に対する数値割り当てをスキップする。成立する場合は、残りの文字に対する数値割当を継続する。M, C, B, P, A, I まで数値割り当てを行うと、全ての文字の数値割当が決まるので、全体の等式

$100\times A - 91\times C + 2\times M - 900\times I + 10\times B - 10\times P = 0$

をチェックし、この割り当てが等式を満たすかどうかを判定する。

- この考え方に基づくプログラム例を以下に示す。Core i7 1.8GHz 上で 4 種類の判定データに適用すると、各々0.5 秒程度で結果が得られる。
- ※　国内予選では、プログラムの実行速度は問われないので、2〜3 秒程度で結果が得られる前述の簡単なプログラムで OK.

```
/*  ACM-ICPC2009 国内予選 Problem C */
// http://www.waseda.jp/assoc-icpc2009/preliminary/contest/all_ja.html#section_C
// filename = pc2.c
// コンパイル： cc -O2 pc2.c
// 実行方法：   ./a.out < C0 > C0.result 等
// 確認方法：   diff C0.ans C0.result 等
// アルゴリズム：覆面算に現れる文字に対して下位の桁に表れるものから順に
//               互いに異なる数値を割当る。
//               最下位桁に表れる文字への数値割当がすんだ時点で
//               覆面算の最下位桁の等式が成立するかをチェックし
//               不成立なら上位桁に対する割当をスキップする。
//               最下位桁の等式が成立するなら、次の文字に対する割当を継続し
//               下位２桁に表れる文字への数値割当がすんだ時点で
//               覆面算の下位２桁の等式が成立するかをチェックする。
//               このような処理を最上位桁まで繰り返す。

#include <stdio.h>
#include <string.h>

#define MAXN 12         // N の最大値
#define MAXLEN 8        // 文字列の長さの最大値

int n;                  // N,  3 <= n <= 12
char strs[MAXN][MAXLEN+1];      // STRING_i を strs[i-1][ ]に格納
int len[MAXN];          // 各 STRING の長さ
char c[10];             // STRING に表れる文字
int v[10];              // 各文字に割当てた値
int a[10][10];          // 各文字の係数. a[i][ ]は下から i 桁目までの等式の係数
                        // ただし、a[0][ ]は全体の等式
int ce[10];     // ce[i] = i 文字目まで割当てた時点でチェックすべき等式番号
int md[10];             // 各文字が表れる最下位の桁位置
int maxlen = 0;
int nc;                 // 使われている文字数
int zskip[10];          // 1 -> 値 0 は割当不可
```

```
// 桁位置の重み
int weight[] = {1,10,100,1000,10000,100000,1000000,10000000};
int count;              // 等式を満たす割当の個数
int nu[10];             // 未使用の数値リスト

// この時点でチェック可能な等式があればチェックする
// 数値を未割当の文字が残っている場合は、未使用の数値を順番に割当てる
void perm(int level)    // level = 数値割当済みの文字数
{
  int i,j;
  int val;
  int zl;
  int *ap,*vp;

  if(level == nc){      // 全ての文字に対して数値割当ずみ
    ap = &a[0][0];      // check する条件式の係数の配列
    val = 0;
    for(i=0; i<nc; i++)
      val += v[i] * (*ap++);
    if(val == 0) count++;
    return;
  }
  if(ce[level] > 0){    // この時点でチェック可能な条件式がある
    ap = &a[ce[level]][0];      // その条件式の係数の配列
    val = 0;
    for(i=0; i<level; i++)
      val += v[i] * (*ap++);
    val = val % weight[md[level]-1];    // 下位 md[level] 桁で判定
    if(val != 0) return;        // 条件不成立なら残っている文字への数値割当不要
  }
  zl = zskip[level];            // zl = 1 なら 0 は割当不可
  for(i=0; i<10-level; i++){    // 未使用の数値を順に割当てていく
    val = nu[i];                        // val は未使用の数値
    if(val==0 && zl) continue;          // 0 が割当不可なら次へ
    v[level] = val;             // val を割当てる
    nu[i] = nu[9-level];        // val は使用済みとする
    perm(level+1);              // 再帰的に次の文字に数値を割当てていく
    nu[i] = val;                // val を未使用に戻す
  }
}

int main()
{
  int i,j,k;

  while(1){
    scanf("%d", &n);    // N を入力
    if(n==0) break;     // N=0 なら終了
    for(i=0; i<n; i++){         // n 個の文字列を読み込む
      scanf("%s", &strs[i][0]);         // 文字列を読み込み
      len[i] = strlen(&strs[i][0]);     // その長さをセット
      if(len[i] > maxlen) maxlen = len[i];      // 文字列の最大長を更新
    }
    nc = 0;
```

```
    for(i=0; i<10; i++){
      c[i] = md[i] = -1;
      zskip[i] = 0;
    }
    for(i=0; i<10; i++)
      for(j=0;j<10; j++)
        a[i][j] = 0;
    for(i=1; i<=8; i++)
      for(j=0; j<n; j++){
        char *s;
        int d,w,lj;
        lj = len[j];
        s = &strs[j][0];
        if(i <= lj){
          d = s[lj-i];
          for(k=0; k<nc; k++){
            if(c[k] == d) break;
          }
          if(k==nc){
            md[k] = i;
            c[nc++] = d;
          }
          if(i == lj && lj > 1) zskip[k] = 1;
          w = weight[i-1];
          if(j == n-1) w *= -1;
          a[i][k] += w; // 下位 i 桁の文字への数値割当決定時の条件式
          a[0][k] += w; // 全ての文字への数値割当決定時の条件式
        }
      }
    // 下位 i 桁以下の文字への数値割当決定時の条件式を求める
    for(i=2; i<= maxlen; i++)
      for(j=0; j<nc; j++)
        a[i][j] += a[i-1][j];
    // i 個の文字への数値割当決定時にチェックすべき条件式の番号を求める
    for(i=1; i<nc; i++)
      if(md[i-1] != md[i])
        ce[i] = md[i]-1;
      else
        ce[i] = 0;
    ce[0] = 0;
    count = 0;
    for(i=0; i<10; i++) nu[i] = i;
    perm(0);
    printf("%d¥n",count);
  }
}
```