[Problem F]　古い記憶

良さそうな方法は思いつかなかった。アイディア募集中！！！
かなり強引に解いている。各判定データに対して、３０秒程度かかる。

元の文章と改ざん文章の関係を考える。ウィルス感染の可能性は高々２回であり、各々の感染では、１文字削除、１文字追加、１文字変更が行われる。削除と追加は双対な関係であるので、改ざん文章に対して、

- 改ざん文章そのもの（実際にはウィルス感染が起こっていなかった）
- 改ざん文章から任意の１文字を削除（元の文章に対しては１文字追加が発生）
- 改ざん文章の任意の１文字を変更（元の文章に対して１文字変更が発生）
- 改ざん文章の任意の場所に任意の文字を追加（元の文章に対しては１文字削除が発生）
- 改ざん文章に対して任意の２文字を削除
- 改ざん文章に対して任意の２文字を変更
- 改ざん文章に対して任意の２文字を任意の位置に追加
- 改ざん文章に対して１文字削除と１文字追加
- 改ざん文章に対して１文字削除と１文字変更
- 改ざん文章に対して１文字変更と１文字追加

により元の文章の候補を生成する。候補の生成において、文字の変更や追加に対しては、該当部分の文字を「?」としておく。
このような「?」を含む文章に対して、文章の各位置が少なくとも１個のピースでカバーされることをチェックする。
「?」を一つも含まない場合は、候補文章に対して、各ピースが部分文字列として現れるかをチェックし、候補文章の全ての位置が少なくとも一つ以上のピースの部分文字列となっていれば候補として採用する。
「?」を含む場合は、「?」ごとに、まず「?」に代入することによりその部分があるピースの部分文字列となる可能性がある文字のリストを求める。その後「?」にそれらの文字を順次代入して、「?」を含まない場合と同様の判定を行う。

なお、同じ候補文字列を複数回チェックするのをさけるため、チェックした文字列は２分探索木に登録しておく。また、最終的に候補となる文章の個数が５以下の場合は、その文章を辞書式順にプリントする必要があるので、候補となることが分かった文章も２分探索木に登録している。

プログラム例

```
// ACM-ICPC 2010 Japan Online Contest Problem F
// http://icpc2010.honiden.nii.ac.jp/domestic-contest/problems#section_F
//      ファイル名: pf.c
//      コンパイル方法: cc pf.c
//      実行方法: ./a.out < F0 > F0.result  など
```

```
//      チェック方法: diff F0.ans F0.result など
//
// アルゴリズム概略
// 改ざん文章に最大 2 回の文字削除/変更/挿入を行ったものが候補文章となりうる
// 変更や挿入に対し、その位置の文字を'?'で表し、これらの操作により得られる
// 文字列を作成。各ピースに対して、そのピースが'?'部分をカーバーできる可能性を
// 探り、可能性がある場合にその'?'が取りうる文字のリストを作成する。
// '?'位置に候補となりうる文字を代入した文字列を作成し、その文字列の各文字が
// 少なくとも一つのピースによりカバーされているかをチェックする。
// なお文字の削除しか行わない場合や、そもそも感染していない場合は、チェック対象の
// 文章に'?'は含まれないので、この文章の各文字が少なくとも一つのピースにより
// カバーされているかどうかを判定すればよい。
// 同じ文字列を繰り返しチェックするのをさけるため、チェック済みの文字列は
// 二分探索木で管理する。また、候補となる文字列も 2 分探索木で管理する。
// 後者の 2 分探索木を in order で出力すれば候補文字列を辞書順に出力することができる。

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

#define MAX_N 30          // ピースの数の最大値
#define MAX_SLEN 42       // 元の文章の長さの最大値 = 改ざん文章の長さの最大値＋ 2
#define MAX_PLEN 20       // ピースの長さの最大値

struct node_type {          // 2 分探索木の節点の構造体
   char *sentence;          // 登録する文字列
   struct node_type *left;      // 左側の子供へのポインタ
   struct node_type *right;   // 右側の子供へのポインタ
};
typedef struct node_type node_t;      // 上記の構造体の型を node_t と名付ける

node_t *candidate;              // 候補となる文章を格納するための 2 分探索木
node_t *checked;                // チェックした文字列を格納するための 2 分探索木

char pieace[MAX_N][MAX_PLEN+1];       // ピース文字列を格納する配列
char original[MAX_SLEN+1];            // 元の文章を格納する配列
char altered[MAX_SLEN+1];             // 改ざん文章を格納する配列
char ans[5][MAX_SLEN+1];              // 可能な元の文章を格納する配列
int   used[128];                      // ピースの中に現れる文字を示す配列
                                      // used[c]==1 ---> 文字 c がピースに含まれる
                                  // used[c]==0 ---> 文字 c はピースに含まれない
char chused[27];                      // ピースに含まれている文字のリスト
char chused1[27],chused2[27];

int d;        // 感染回数の最大値
int n;        // ピースの数
int nc;       // ピースに現れる文字の種類数
int c;        // もとの文章の候補数
int nc1, nc2;

void clean_bst(node_t *pt);
```

```
int print_bst(node_t *pt, int count);
void check0(char *str);
void gen_candc(char *str, int p1, int p2);
void check(char *str);
void check_del(char *str, int level);
void check_chg(char *str, int level);
void check_ins(char *str, int level);
void solve();

// 2 分探索木に文字列を登録する
// str =  登録する文字列, ppt = 2 分探索木の根節点へのポインタを格納している場所
// mode = 1 ---> 候補となる文字列の登録
//  戻り値 :  0 ---> 新規登録成功        1 ---> 既に登録されていた
int ins_bst(char *str, node_t **ppt, int mode)
{
  node_t *pt;
  int sc;
  node_t *root;

  root = *ppt;                              // root -> 2 分探索木の根節点
  if(root == NULL){                          // 2 分探索木が空の場合
    root = (node_t *)malloc(sizeof(node_t));      // 2 分探索木の節点を格納する領域を確保
    root->left = root->right = NULL;      //  この新しい節点は左右の子を持たない
    root->sentence = (char *)malloc(strlen(str)+1);
                                            //  この節点に文字列を格納する領域を確保
    strcpy(root->sentence, str);           //  確保した領域に文字列をコピー
    if(mode) c++;                           //  候補文字列を登録した場合は候補数を 1 増やす
    *ppt = root;                           //  この節点を 2 分探索木の根節点とする
    return 0;                               //  新規登録成功
  }
  pt = root;                               // pt -> 2 分探索木の根節点
  while(pt){                                //  pt  が NULL で無い間以下を繰り返す
    sc = strcmp(str, pt->sentence);      //  今回登録する文字列と
                                            //  pt が指す節点に登録されている文字列を比較
    if(sc < 0){              //  今回登録する文字列の方が小さく
      if(pt->left)          //  左の子が存在するなら
        pt = pt->left;    //  pt  を左の子へのポインタとする
      else break;           //  左の子が存在しない場合は、繰り返しを終了
    }
    else if (sc == 0){    //  今回登録する文字列がこの節点に登録済なら
      return 1;              //   1 を返す(登録済)
    }
    else {                    //  今回登録する文字列の方が大きく
      if(pt->right)        //  右側の子が存在するなら
        pt = pt->right; //  pt  を右の子へのポインタとする
      else break;           //  右の子が存在しない場合は、繰り返しを終了
    }
  }
  if(sc<0){                    //  今回登録する文字列の方が小さく左の子が存在しない場合
    pt->left = (node_t *)malloc(sizeof(node_t));      //  左の子を格納する領域を確保し
    pt = pt->left;          //  pt を左の子へのポインタとする
  }
```

```
  else {                        // 今回登録する文字列の方が大きく右の子が存在しない場合
    pt->right = (node_t *)malloc(sizeof(node_t));    // 右の子を格納する領域を確保し
    pt = pt->right;       // pt を右の子へのポインタとする
  }
  pt->left = pt->right = NULL;  // pt が指す新しい節点は左右の子は持たない
  pt->sentence = (char *)malloc(strlen(str)+1); // その節点に登録する文字列を格納する
                                                  // 領域を確保
  strcpy(pt->sentence, str);     // 確保した領域へ文字列をコピー
  if(mode) c++;              // 候補文字列の登録なら、候補数を 1 増やす
  return 0;                  // 新規登録成功
}

void clean_bst(node_t *pt)    // 2 分探索木が使用していた領域を解放
{
  if(pt){
    free(pt->sentence);       // pt が指す節点の文字列を格納していた領域を解放
    clean_bst(pt->left);      // 左の子以下を再帰的に post order 順に解放
    clean_bst(pt->right);     // 右の子以下を再帰的に post order 順に解放
    free(pt);                 // 最後にこの節点自体の領域を解放(post order)
  }
}

// pt が指す節点を根節点とする 2 分探索木に登録されている文字列を
// count 個辞書順(inorder)に印刷。戻り値は残り印刷個数。
int print_bst(node_t *pt, int count)
{
  int pn;
  if(count==0) return 0;            // count が 0 なら何もしない
  if(pt==NULL) return count;      // pt が NULL なら count を返す
  count = print_bst(pt->left, count);    // まず左の子以下を再帰的に in order で
                                          // count 個印刷
  if(count == 0) return 0;         // 残り印刷数が 0 なら終了
  printf("%s¥n", pt->sentence); // この節点に登録されている文字列を印刷
  count--;                          // count を 1 減らす
  count = print_bst(pt->right,count);    // 右の子以下を再帰的に in order で
                                          // count 個印刷
  return count;                     // 残り印刷数を返す
}

// 文字列 str が、元の文章の候補となりうるか判定し、
// 候補になる場合は、候補文字列を格納する二分探索木に登録し
// 候補数を 1 増やす
void check0(char *str)
{
  int cover[MAX_SLEN];  // 文字列 str の各文字がピース文字列により
                        // カバーされているかを調べるための作業用配列
  int i,j,len,lenp;
  char *cp, *pos;

  len = strlen(str);      // 文字列 str の長さ
  for(i=0; i<len; i++) cover[i] = 0;      // 作業用配列を 0 で初期化
                                          // （どの文字もカバーされていない）
```

```
  for(i=0; i<n; i++){
    cp = str;                                  // cp: 最初は与えられた文字列(str)全体
    lenp = strlen(pieace[i]);                  // i 番目のピース文字列の長さ
    while(*cp){                                // cp が NULL で無い間以下を繰り返す
      pos = strstr(cp, pieace[i]);  // 文字列 cp の中に i 番目のピースが部分文字列として
      if(pos == NULL) break;                   // 含まれていなければ、
                                               // i 番目のピースに対する処理終了
      for(j=pos-str; j<pos-str+lenp; j++) cover[j]=1;
                                               // i 番目のピースに一致する部分を 1 にセット
      cp = pos+1;      // 一致した位置の次の場所から再度一致するかを繰り返しチェック
    }
  }
  for(i=0; i<len; i++)
    if(cover[i] == 0) return;    // カバーされていない文字が 1 文字でもあれば終了
  ins_bst(str, &candidate, 1);  // カバーされているので、
                                // 候補文字列として二分探索木に登録
}

// '?'の位置(最大 2 か所）に入りうる文字のリストを作成
// p1 は最初の'?'の位置、p2 は 2 番目の'?'の位置
void gen_candc(char *str, int p1, int p2)
{
  int used1[128], used2[128];    // used1[c]==1 ---> 最初の'?'の位置に文字 c が入りうる
                                 // used2[c]==1 ---> 二番目の'?'の位置に文字 c が入りうる
  int i,j,k;
  int lens, lenp;
  for(i=0; i<128; i++)
    used1[i] = used2[i] = 0;     // 配列 used1 と used2 を 0 で初期化
  lens = strlen(str);            // lens == 文字列 str の長さ
  for(i=0; i<n; i++){            // 各ピースに対して
    char *cp;
    cp = pieace[i];              // cp = i 番目のピース
    lenp = strlen(cp);           // lenp = i 番目のピースの長さ
    for(j=0; j<=lens-lenp; j++){
      char q1, q2;
      q1 = q2 = 0;
      for(k=0; k<lenp; k++){
        if(str[j+k] == cp[k]) continue;
        if(str[j+k] == '?') {    // str の j 番目から'?'の手前まで i 番目のピースと一致
          if(j+k == p1) q1=cp[k];    // 最初の'?'なら
                                     // その位置に対応するピースの文字を q1 に
          else q2=cp[k];             // 2 番目なら q2 に入れる
          continue;
        }
        else break;                  // 一致しない場合はそのピースに対する処理終了
      }
      if(k==lenp){                   // ピースの最後まで一致したら
        if(q1) used1[q1] = 1;        // q1 を最初の'?'に利用できる文字としてマーク
        if(q2) used2[q2] = 1;        // q2 を二番目の'?'に利用できる文字としてマーク
      }
    }
  }
```

```
  // 利用可としてマーク付けされた文字をリストの形に変換
  j = k = 0;
  for(i='.'; i<='Z'; i++){
    if(used1[i]) chused1[k++]=i;      // 最初の'?'に利用できる文字のリストを構成
    if(used2[i]) chused2[j++]=i;      // 二番目の'?'に利用できる文字のリストを構成
  }
  nc1 = k;  // 最初の'?'に利用できる文字のリストの長さ
  nc2 = j;  // 二番目の'?'に利用できる文字のリストの長さ
}

// '?'を含む(最大 2 か所)文字列が候補文章となる可能性を判定
void check(char *str)
{
  char work[MAX_SLEN+1];
  int i,j,len;
  int nq, q1, q2;
  int c1,c2;

  if(ins_bst(str, &checked,0)) return;  // 既にこの文字列をチェック済みなら何もしない
  len = strlen(str);            // len =  文字列の長さ
  nq = 0;                       // nq  は  '?'の個数
  q1 = q2 = -1;                 // '?'の位置を見つける
  for(i=0; i<len; i++){
    if(str[i] == '?'){          // '?'が見つかったので
      nq++;                     // nq  を 1 増やす
      if(nq==1) q1=i;           // 最初の'?'ならその位置を q1 に
      else {
        q2 = i;                 // 2 回目の'?'ならその位置を q2 に格納
        break;
      }
    }
  }
  switch(nq){
  case 0:                       // '?'が無い場合
    check0(str);      // '?'を含まない文字列のチェックを呼び出す
    break;
  case 1:                       // '?'が 1 個ある場合
    gen_candc(str, q1, q2);     // '?'の位置に入る可能性がある文字のリストを作成
    c1 = str[q1];     // '?'を退避
    for(i=0; i<nc1; i++){
      str[q1] = chused1[i];     // '?'の位置に可能性のある文字を順次代入し
      check0(str);              // 候補文章となるかどうかを判定
    }
    str[q1] = c1;     // q1  番目の文字を'?'に戻す
    break;
  case 2:                       // '?'が 2 個ある場合(場所は q1 と q2)
    gen_candc(str, q1, q2);     // '?'の位置に入る可能性がある文字のリストを作成
    c1 = str[q1];     // '?'を退避
    c2 = str[q2];     // '?'を退避
    for(i=0; i<nc1; i++){
      str[q1] = chused1[i];     // 最初の'?'の位置に可能性のある文字を順次代入し
      for(j=0; j<nc2; j++){
```

```
            str[q2] = chused2[j];     // 更に二番目の'?'の位置にも
                                      // 可能性のある文字を順次代入し
            check0(str);              // 候補文章となるかどうかを判定
          }
        }
        str[q1] = c1;          // q1 番目の文字を'?'に戻す
        str[q2] = c2;          // q2 番目の文字を'?'に戻す
    }
}

// 文字を削除した文章が候補文章となる可能性を判定
// level==1 ---> まず 1 文字削除した文章の可能性判定を行い、
//                  d==2 なら、更にもう 1 文字削除/変更/挿入した場合の可能性も判定
// level==2 ---> 1 文字削除した文章の可能性を判定
//                  (既に 1 文字削除した状態で呼び出される)
void check_del(char *str, int level)
{
    int i,j,len;
    char *cp;
    char work[MAX_SLEN+1];          // 文字を削除するために使用する作業用配列
    len = strlen(str);              // len = 文字列の長さ
    for(i=0; i<len; i++){           // 各 i = 0 ～ len-1 に対して
        cp = work;
        for(j=0; j<len; j++){       // i 番目の文字を削除した文字列を work に作成し
            if(j==i) continue;
            *cp++ = str[j];
        }
        *cp = 0;
        check(work);                // それが候補文章となる可能性を判定
        if(d==2 && level==1){       // もし感染の可能性が 2 回あり、今の削除が 1 回目なら
            check_del(work, 2);     // もう 1 回削除するケースと
            check_chg(work, 2);     // 変更するケースと
            check_ins(work, 2);     // 挿入するケースについて可能性を判定
        }
    }
}

// 文字を変更した文章が候補文章となる可能性を判定
// level==1 ---> まず 1 文字変更した文章の可能性判定を行い、
//                  d==2 なら、更にもう 1 文字変更/挿入した場合の可能性も判定
// level==2 ---> 1 文字変更した文章の可能性を判定
//                  (すでに 1 文字削除/変更した状態で呼び出される)
void check_chg(char *str, int level)
{
    int i,j,k,len;
    char *cp, work[MAX_SLEN+1];     // 文字を変更するために使用する作業用配列
    char cc;
    len = strlen(str);              // len = 文字列の長さ
    strcpy(work,str);               // まず作業用配列へ文字列をコピー
    for(i=0; i<len; i++){           // 各 i = 0 ～ len-1 に対して
        cc = work[i];               // i 番目の文字を cc に退避し
        work[i] = '?';              // その位置の文字を'?'として
```

```
      check(work);                    // その文字列が候補文章となる可能性を判定
      if(d==2 && level==1){           // もし感染の可能性が 2 回あり今の変更が 1 回目なら
        check_chg(work, 2);           // もう 1 回変更するケースと
        check_ins(work, 2);           // 1 文字挿入するケースについて可能性を判定
      }
      work[i] = cc;                   // i 番目の文字を元の文字に戻す
    }
  }

  // 文字を挿入した文章が候補文章となる可能性を判定
  // level==1 ---> まず 1 文字挿入した文章の可能性判定を行い
  //                d==2 ならさらにもう 1 文字挿入した文章の可能性も判定
  // level==2 ---> 1 文字挿入した文章の可能性を判定
  //                （既に 1 文字削除/変更/挿入した状態で呼び出される）
  void check_ins(char *str, int level)
  {
    int i,j,k,len;
    char *cp;
    char work[MAX_SLEN+1];          // 文字を挿入するために使用する作業用配列
    len = strlen(str);              // len = 文字列の長さ
    for(i=0; i<=len; i++){          // 各 i = 0 ～ len に対して
      k=0;
      for(j=0; j<=len; j++){        // i 番目に 1 文字(?)挿入した文字列を作成し
        if(i==j) work[j] = '?';
        else work[j] = str[k++];
      }
      work[len+1] = 0;
      check(work);                  // その文字列が候補文章となる可能性を判定
      if(d==2 && level==1)          // もし感染の可能性が 2 回あり今挿入したのが 1 回目なら
        check_ins(work, 2);         // level=2 で再度 check_ins を呼び出す
    }
  }

  void solve()
  {
    int i,j;
    char *cp;
    for(i=0; i<128; i++) used[i]=0;   // ピース中に現れる文字を示す配列を初期化
    for(i=0; i<n; i++){               // 各ピースに対して
      cp = pieace[i];                 // cp -> ピース文字列
      while(*cp)                      // ピース文字列の各文字に対して
        used[*cp++] = 1;              // 配列 used のその文字に対応する位置に 1 を代入
    }
    nc = 0;                           // nc はピースに現れる文字の種類数
    for(i=0; i<128; i++)
      if(used[i]) chused[nc++] = i;   // 配列 chused にピースに現れる文字のリストを構築
    c = 0;                            // 元の文章の可能性の数を 0 に初期化
    candidate = checked = NULL;       // 候補とチェック済みの文字列を格納する
                                      // 空の 2 分探索木を作成
    check(altered);                   // ウイルスに 1 度も感染しなかったケースがありうるかチェック
```

```
  check_del(altered, 1);      // 改ざん文章から 1 文字消去した文章の可能性をチェック
  check_chg(altered, 1);      // 改ざん文章に対し 1 文字変更した文章の可能性をチェック
  check_ins(altered, 1);      // 改ざん文章に 1 文字挿入した文章の可能性をチェック
  printf("%d¥n",c);             // 候補の数を出力
  if(c <= 5) print_bst(candidate, c);    // それが 5 以下の場合は、候補を辞書順で出力
  clean_bst(checked);           // チェック済み文字列を格納していた二分探索木の領域を
解放
  clean_bst(candidate);       // 候補の文字列を格納していた二分探索木の領域を解放
}

int main()
{
  int i;
  while(1){
    scanf("%d %d", &d, &n);             // d と n の入力
    if(d==0 && n==0) break;             // 両方とも 0 なら終了
    scanf("%s",altered);                // 改ざん文章の入力
    for(i=0; i<n; i++)
      scanf("%s", pieace[i]);           // ピースを n 個入力
    solve();                            // 解を求める
  }
}
```