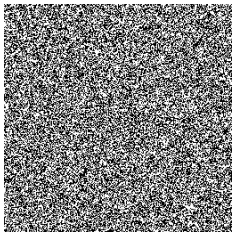
課題と問題の詳細説明

暗号化された画像の解読

課題概要で予告したように，今回の本選課題は，暗号化された画像の解読です．具体的には，256×256 の白黒画素 (black(0)/white(1) pixel) が画素ごとに暗号化され，たとえば，下図（左）のようなランダムに見える画像が与えられます．それに対し，各画素ごとに暗号を解き，元の画像（たとえば，下図（右））のような画像を求めよ，というのが今回の課題です．ただし，左の画像だけからの解読は非常に難しいので，各画素ごとに解読のヒントとなる情報も与えることにします．



解 読
$\Longrightarrow$
$\Longleftarrow$
暗号化

これらの情報は，暗号化された画像のファイル `eimagefile` とヒント情報のファイル `hintfile` の 2 つのファイルとして与えられます．

以上が，課題の概要ですが，問題の正確な記述は以下のようになります．

スーパーコン０４の問題

暗号化された画像のファイル `eimagefile`，ヒント情報のファイル `hintfile` に与えられるデータをもとに，できる限り正確な元画像を求めるプログラムを作成せよ．

プログラムの性能は，評価用に用意された 1 組のファイルに対し，制限時間（3 分）以内に，出力された画像の正解率（元画像と一致する画素数の割合）で評価する．ただし，同じ正解率の画像を出力した場合には，出力までの計算時間が短いものを良いプログラムとする．

（注：制限時間内であれば，何度でも画像を出力してよい．複数出力された場合には，最後に出力された画像を用いる．）

以下では，１．暗号化法，２．ヒント情報，３．データの形式と入出力法，４．利用できるツールについて，順に説明していきます．

１．暗号化法

画像の暗号化には，通常の RSA 暗号方式を使う．もっと具体的には，1 つの画像の暗号化のためには，各 100 ビット以下の数からなる RSA セット (n, d, e) が 1 つ用いられる（RSA 暗号の基本については，付録の補足説明を参照）．このうち，公開鍵 n と e は，ヒント情報の一部として与えられるので，それを用いれば，この RSA セットに対する暗号化関数 $e_{\rm rsa}$ は容易に計算可能．一方，秘密鍵 d は与えられないので，この RSA セットを破らない限り，復号化関数 $d_{\rm rsa}$ の計算は難しい．

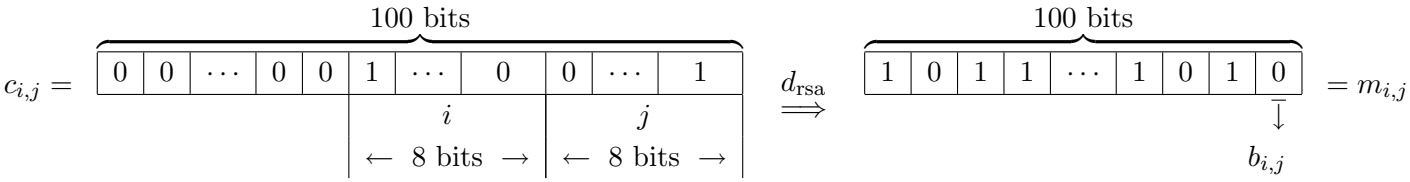
関数 $e_{\rm rsa}$, $d_{\rm rsa}$ は，mod n 上で計算されるので，定義域，値域は，ともに $\{0, ..., n-1\}$ だが，以下では簡単に，各関数に与えられる数，それらが計算する数は，それぞれ 100 ビットの整数と考えることにする．

さて，画像の暗号化について述べよう．画像は 256×256 の白黒画素だが，ここでは説明のため，その画素情報が配列 im[256][256] に入っているものとする．つまり，各 (i, j), $0 \le i, j \le 255$, に対し，im$[i][j]$ は上から i 番目，左から j 番目の画素 (0 or 1) を表わす．一方，eim[256][256] は，暗号化された画素情報が入る配列とする．以上のもと，各 (i, j) の画素に対する暗号化の計算は次のように記述できる．

$$
\begin{array}{lcl}
c_{i,j} & = & i \times 256 + j \\
m_{i,j} & = & d_{\rm rsa}(c_{i,j}) \\
b_{i,j} & = & m_{i,j} \text{ の最下位ビット (least significant bit)} \\
\texttt{eim}[i][j] & = & \texttt{im}[i][j] \oplus b_{i,j} \quad \text{（注：演算 } \oplus \text{ は排他論理和）}
\end{array}
$$

２．ヒント情報

復号化関数 $d_{\rm rsa}$ は 100 ビット整数を 100 ビット整数に変換する関数なので，上記の暗号化操作をイメージ的に表わすと次のようになる．

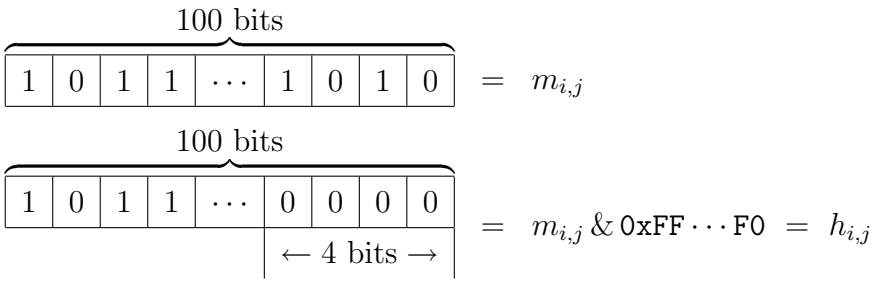


画像の暗号化は，このように計算されるので，復号化は，各 (i, j) に対して $b_{i,j}$ を計算することに他ならない．これは，$d_{\rm rsa}$ を知っていれば，もちろん，簡単だが，それには与えられた RSA セットを破らなければならない．

一方，$e_{\rm rsa}$ ならば計算できるので，$d_{\rm rsa}$ を知らなくても $b_{i,j}$ を求めることは可能である．すべての $m_{i,j}$ の候補 m に対し，$e_{\rm rsa}(m)$ を求め，ちょうど $c_{i,j}$ となる m を $m_{i,j}$ とすればよい．しかし 2^{100} 個の候補（正確には n 個の候補）について，すべてを調べるのは，非常に困難だろう．そこで今回は，各 (i, j) に対して，$m_{i,j}$ の右 h ビットだけを 0 にマスクした

情報（つまり，$m_{i,j}$ の上位 $100-h$ ビットの情報）を，画素 (i,j) に対するヒント情報 $h_{i,j}$ として与えることにする．

たとえば，$h=4$ ビットをマスクする場合には，次のような情報がヒントとして与えられることになる．



こうしたヒントが与えられていれば，マスクされた h ビットを求めればよく，それには，すべてを試してみればよいことになる．（ちなみに，その計算には e_{rsa} を，最悪で 2^h 回実行しなければならない．そこで h をヒントの困難さ (hardness) と呼ぶことにする．）

３．データの形式と入出力法

入力データは，暗号化された画像がファイル `eimagefile` に，ヒント情報がファイル `hintfile` に，以下のような形式で与えられる．なお，各ヒント $h_{i,j}$ は，32 桁（128 ビット）の 16 進数．（本大会を通し，RSA の計算は，すべて 32 桁 16 進数上の計算とする．）

ファイル `eimagefile` の例

`eim`$[i][j]$ を $i=0$ から各行ごと，$j=0$ から順に

```
0 1 1 0 1 0 1 ···1 0 1 1 0
0 1 0 0 0 1 0 ···0 0 0 0 1
  ⋮
```

ファイル `hintfile` の例

最初の行はコメント，2, 3 行目は RSA 情報，以下，各画素とごに $c_{i,j}$, h, $h_{i,j}$ の 3 つのデータ．なお，`#` より右はコメント．

```
# hint for test data 1 (easy)
000000001BC8B211 4C4798F5F3E5A6EAF  # n（16 進数），16 桁で空白が入る
B # e（16 進数），特例として 1 桁で表わす
0   0  0000000000000000 0000000000000000  # (0,0) のヒント
1   0  0000000000000000 0000000000000000  # (0,1) のヒント
2 30  000000007477FDB6 99D22F5A00000000  # (0,2) のヒント
  ⋮
65535 15  00000000DB23889F C2EE8CC8E2018000 # (255,255) のヒント
```

一方，答えの画像データは，暗号化画像ファイル eimagefile と同じ形式のものを標準出力に出して欲しい．ただし，審査で出力時刻を確認する必要があるので，出力時刻付きで標準出力に出すための手続き outresult を必ず使用すること．使用するには，配列 int image[65536] を用意し，各 (i, j) における元画素（の予測）im$[i][j]$ を以下のように格納して呼び出す．

```
for(i = 0; i <= 255; i++)
   for(j = 0; j <= 255; j++) image[i*256+j] = im[i][j];
outresult(image)
```

なお，制限時間内であれば，何度でも outresult を利用して出力することが可能．ただし，複数出力された場合には，時間内で 最後に出力されたもの を審査の対象とする．また，実行開始から審査された画像が出力されるまでの時間が計算時間となる．

４．補助として利用できる関数について

プログラミングを支援するために，今回の課題に必要な基本関数群を用意した．これは共通のヘッダーファイル supercon.h でプロトタイプ宣言され，その定義（supercon.c に記述）は，supercon.o と一緒にコンパイルすることで取り込まれる．このヘッダーファイルでは，標準的なヘッダーファイルの取り込みも行う．プログラム中では，この共通のヘッダーファイル supercon.h 以外には読み込まないこと．

以下に supercon.h について簡単に説明しておく．詳しくは直接 supercon.h, supercon.c を読んで欲しい．

supercon.h の中で取り込まれるヘッダーファイル

stdio.h, stdlib.h, string.h, mpi.h

bignum 型，補助の関数

- 128 ビット整数のための bignum 型

```
typedef struct bignum {
 unsigned long l;  （注：unsigned long は 64 ビット整数）
 unsigned long h;
} bignum;
```

- 128 ビット整数上の mod 演算

```
bignum madd(bignum x, bignum y, bignum n);      (x+y)%n
bignum msub(bignum x, bignum y, bignum n);      (x-y)%n
bignum mmul(bignum x, bignum y, bignum n);      (x*y)%n
```

- 128 ビット整数の比較（注：bool は false (0) or true (1) の値を取る型）

```
bool blt(bignum x, bignum y);     x < y (Less Than)
```

```
bool ble(bignum x, bignum y);     x <= y (Less than or Equal to)
bool bgt(bignum x, bignum y);     x > y (Greater Than)
bool bge(bignum x, bignum y);     x >= y (Greater than or Equal to)
bool beq(bignum x, bignum y);     x == y (EQual to)
bool bne(bignum x, bignum y);     x != y (Not Equal to)
```

- 128 ビット整数上の演算（注：扱える数は正整数のみ．桁あふれは無視される）

```
bignum badd(bignum x, bignum y); x + y
bignum bsub(bignum x, bignum y); x − y
bignum bmul(bignum x, bignum y); x * y
bignum bquo(bignum x, bignum y); x / y
bignum brem(bignum x, bignum y); x % y
void bdiv(bignum x, bignum y, bignum * q, bignum * r);
```

- 情報の入出力

```
int getimage(FILE *fp, int eimage[]);   暗号化された画像情報の取り込み
int gethint(FILE *fp, bignum *rsaN, bignum *rsaE, int hardness[], hint[]);
                                        ヒント情報の取り込み
void outstarttime(void);                審査用のコマンド（開始時に必ず実行）
void outresult(int image[]);            復元画像の標準出力への出力
```

注 1. getimage, gethint は，失敗時には 0，正常終了時には 1 を返す

2. 各配列は次のように定義しておく．ただし Nxy は総画素数 65536 .

```
int image[Nxy], eimage[Nxy];
int hardness[Nxy]; bignum hint[Nxy];
```

- 128 ビット整数の入出力（注：デバッグ用）

```
void getnum16(bignum *x);    x の読み込み（16 進で）
void getnum10(bignum *x);    x の読み込み（10 進で）
void putnum16(bignum x);     x の出力（16 進で）
void putnum10(bignum x);     x の出力（10 進で）
bignum btransnum(unsigned long h, unsigned long l);
                                        10 進で 19 桁，19 桁の数を bignum に変換
```

注：bignum の最大値は 3402823669209384634 8034375210639556607（10 進 38 桁）

なお，supercon.o は，各関数のプログラムは supercon.c を最も高い最適化のレベルでコンパイルしたものである．以上の supercon.h, supercon.c, supercon.o は変更しないこと．ただし，outstarttime, outresult 以外の補助関数については，代わりとなる関数を自分たちで新たに作成し，それらを使用しても構わない.（作成した関数は提出するソースファイル中に記述すること）.

付録：RSA 暗号方式について（ごく基本的なことのみ）

RSA 暗号方式とは公開鍵暗号方式の 1 つであり，整数として符号化された文（平文 (plain text) という）を整数の暗号文 (cipher text) に変換する．

RSA 暗号方式における 1 つの RSA 暗号（この説明では誤解を避けるため RSA セット (RSA instance) と呼ぶことにする）は，3 つの数の組 (n, d, e) により定まる．このうち，d を復号鍵もしくは秘密鍵といい，n と e を暗号鍵もしくは公開鍵という．この 3 つの数の背後には，大きな 2 つの素数 p, q があり，次のような関係が成り立つ．

$$\begin{aligned} n &= p \times q, \\ \phi &= (p-1) \times (q-1), \\ d \times e &\equiv 1 \pmod{\phi}. \end{aligned}$$

これだけでは d, e は一意に定まらないが，我々の問題では $e = 11$（$= \texttt{0xB}$）に固定するので，それに対する d は，法 n のもとでは 1 つに定まる．また，p, q には，50 ビット以下の整数を選ぶので，n は 100 ビット以下の整数になる．

RSA セットが決まると，暗号文を計算する暗号化関数，暗号文から平文を求める復号化関数を計算することができる．これらは，以下のように定義される．

$$\begin{aligned} \text{暗号化関数：}\ e_{\mathrm{rsa}}(m) &= m^e \pmod{n}, \\ \text{復号化関数：}\ d_{\mathrm{rsa}}(c) &= c^d \pmod{n}. \end{aligned}$$

当然，$d_{\mathrm{rsa}}(e_{\mathrm{rsa}}(m)) = m$ が成り立つが，RSA 暗号方式の特徴として $e_{\mathrm{rsa}}(d_{\mathrm{rsa}}(c)) = c$ も成立する．

RSA 暗号を破る

RSA 暗号方式を利用する際には，普通，RSA セット (n, d, e) のうち，n と e のみが公開される．n を素因数 p, q に分解できれば，それから ϕ や d を求めることができる．これを RSA 暗号セットを破るという．

RSA 暗号の弱点 !?（我々の問題を解く鍵 !?）

RSA 暗号方式で使われる復号化関数には，次のような特徴がある．そのため，たとえば c_1, c_2 が復号化できると，$c_3 = c_1 \times c_2$ も復号化できることになる．

$$\left.\begin{aligned} d_{\mathrm{rsa}}(c_1) &= m_1 \\ d_{\mathrm{rsa}}(c_2) &= m_2 \end{aligned}\right\} \ \Rightarrow\ d_{\mathrm{rsa}}(c_1 \times c_2) \equiv m_1 \times m_2 \bmod n.$$

一般には，これは大きな問題にはならないが，今回のように特殊な使い方をしているときには，この性質が暗号の弱さを作り出してしまうこともある．実際，我々の課題に対する攻略法を設計する上で役に立つのでは？ よく吟味してみよう．