Épreuve d'informatique de l'X - 2009 - MP/PC

Chiffrement par blocs

Remarque : `quo`, `rem` et `xor` sont les fonctions `iquo`, `irem` et `Xor`([1]) de Maple.

Question 1. Initialement, on a $h = k = \sum_{j=0}^{N-1} a_j N^j$. À la i-ème étape de la boucle, la variable `h` est égale à $\sum_{j=i}^{N-1} a_j N^{j-i}$ et donc le reste de la division par N est a_i que l'on place bien en i-ème position dans le tableau résultat `res` à la ligne 6. À la ligne 7, on modifie `h` en $\sum_{j=i+1}^{N-1} a_j N^{j-i-1}$.

```
1  DecomposerBase := proc(N::posint, k::posint)
2  local i, res, h;
3  res := array(0..N-1);
4  h := k;
5  for i from 0 to N-1 do
6    res[i] := mod(h, N);
7    h := quo(h, N)
8  od;
9  return(res);
10 end;
```

Question 2. À la i-ème étape de la boucle, la variable `h` est égale à $\sum_{k=i}^{N-1} a_k \frac{k!}{i!}$ et donc $h - a_i$ est divisible par $(i+1)$ et a_i est le reste de la division euclidienne par $(i+1)$. On place bien a_i en i-ème position dans le tableau résultat `res` à la ligne 6. À la ligne 7 modifie `kk` en $\sum_{k=i+1}^{N-1} a_k k!/(i+1)!$.

```
1  DecomposerFact := proc(N,k)
2  local h, i, res;
3  res := array(0..N-1);
4  h := k;
5  for i from 0 to N-1 do
6      res[i] := rem(h,i+1);
7      h := quo(h,i+1);
8  od;
9  return(res);
10 end;
```

1. `Xor` est disponible dans Maple 13 à conditions de charger la bibliothèque ≪Bits≫.

Question 3 : On supprime le j-ème élément en faisant attention au fait que les tableaux sont indexés à partir de 0.

```
1  Retirer := proc (L, l, j)
2  local i, res;
3  res := array(0..l-2);
4  for i from 0 to l-2 do
5      if i < j-1
6          then res[i] := L[i]
7          else res[i] := L[i+1]
8      fi;
9  od;
10 return(res);
11 end;
```

Question 4 : On détermine le tableau représentant la permutation σ en suivant l'algorithme donné par l'énoncé. On commence par créer le tableau représentant la liste initiale $[0, 1, \ldots, N-1]$ (ligne 4 à 7). On récupère la décomposition factorielle de k (ligne 8). On applique l'algorithme dans la boucle des lignes 9 à 13.

```
1  EcrirePermutation := proc (N, k)
2  local i, j, sigma, L, decomp;
3  sigma := array(0..N-1);
4  L := array(0..N-1);
5  for i from 0 to N-1 do
6      L[i] := i
7  od;
8  decomp := DecomposerFact(N, k);
9  for i from N-1 by -1 to 0 do
10     j := decomp[i];
11     sigma[N-1-i] := L[j];
12     L := Retirer(L, i+1, j+1)
13 od;
14 return(sigma);
15 end;
```

Question 5 : Pour le déchiffrage d'un bloc b, on recherche son antécédent par σ, ce qui se fait dans la boucle `while` des lignes 11 à 13.

```
1  Chiffrer := proc (N, k, b)
2  local sigma;
3  sigma := EcrirePermutation(N, k);
4  return(sigma[b])
5  end;
6
7  Dechiffrer := proc (N, k, b)
8  local sigma, i;
9  sigma := EcrirePermutation(N, k);
10 i := 0;
11 while sigma[i] <> b do
12     i := i+1
13 od;
14 return(i);
15 end;
```

II. Réseau de Feistel

Question 6 : On crée une variable globale `deuxpuiss32` égale à 2^{32}.

```
1  deuxpuiss32 := 4294967296;
2
3  FeistelTour := proc (k, b)
4  local q, r;
5  r := rem(b, deuxpuiss32);
6  q := quo(b, deuxpuiss32);
7  return( deuxpuiss32.r+Xor(q, F(k, r)));
8  end;
```

Question 7 : Pour décoder, on utilise la propriété $\text{xor}(\text{xor}(b, x), x) = b$ et l'égalité $r_i = q_{i+1}$. Ceci qui permet de calculer q_i à partir de r_{i+1} et q_{i+1}.

```
1  FeistelInverseTour := proc (k, b)
2  local q, r, tmp;
3  tmp := rem(b, deuxpuiss32);
4  r := quo(b, deuxpuiss32);
5  q := Xor(tmp, F(k, r));
6  return(deuxpuiss32*q+r);
7  end;
```

Question 8 : Le tableau K étant supposé indexé à partir de 0, on exécute itérativement les différents tours pour les clefs contenues dans K (boucle des lignes 4 à 6).

```
1  Feistel := proc (K, l, b)
2  local res, i;
3  res := b;
4  for i from 0 to l-1 do
5      res := FeistelTour(K[i], res)
6  od;
7  return(res);
8  end;
```

Question 9 : Ici on décode à l'aide de la fonction `FeistelInverseTour` en utilisant les différentes clefs en partant de la fin (boucles des lignes 4 à 6).

```
1  FeistelInverse := proc (K, l, b)
2  local res, i;
3  res := b;
4  for i from l-1 by -1 to 0 do
5      res := FeistelInverseTour(K[i], res)
6  od;
7  return(res);
8  end;
```

III. Vérification de propriétés statistiques

Question 10 : On construit le tableau S_n. Pour i variant de 0 à $\frac{n}{64}-1$ (boucle des lignes 5 à 11), on décompose $\sigma(i)$ en binaire en rangeant les bits dans le tableau de résultat `res` (boucle des lignes 5 à 10). On faisant attention de les ranger dans le bon ordre (bit de poids fort à gauche).

```
1  sequence := proc (n)
2  local p, i, res, s, k;
3  res := array(0..n-1);
4  p := n/64;
5  for i from 0 to p-1 do
6      s := Sigma(i);
7      for k from 63 by -1 to 0 do
8          res[64*i+k] := rem(s, 2);
9          s := quo(s, 2)
10     od;
11 od;
12 return(res);
13 end;
```

Question 11 : On comptabilise les 0 et les 1 de la séquence S_n (boucle des lignes 5 à 10) dans les variables `n0` et `n1`. On calcule enfin V_1 ligne 11.

```
1  CalculerV1 := proc (n)
2  local Sn, n0, n1, i, res;
3  Sn := sequence(n);
4  n0 := 0; n1 := 0;
5  for i from 0 to n-1 do
6     if Sn[i] = 1
7        then n1 := n1+1
8        else n0 := n0+1
9     fi;
10 od;
11 res := (n1-n0)*(n1-n0)/n;
12 return(evalf(res));
13 end;
```

Question 12 : On comptabilise les occurrences des séquences `00`, `01`, `10` et `11` dans les variables `n00`, `n01`, `n10` et `n11` dans la boucles des lignes 9 à 16 ainsi que les occurrences de 0 et de 1 dans les variables `n0` et `n1`.

```
1  CalculerV2 := proc (n)
2  local n0, n1, Sn, n00, n01, n10, n11, i, res;
3  Sn := sequence(n);
4  if Sn[0] = 1
5     then n0 := 0; n1 := 1
6     else n0 := 1; n1 := 0
7  fi;
8  n00 := 0; n01 := 0; n10 := 0; n11 := 0;
9  for i from 0 to n-2 do
10       if (Sn[i] = 1 and Sn[i+1] = 1) then n11 := n11+1
11    elif (Sn[i] = 1 and Sn[i+1] = 0) then n10 := n10+1
12    elif (Sn[i] = 0 and Sn[i+1] = 1) then n01 := n01+1
13    else n00 := n00+1
14      fi;
15      if Sn[i+1] = 0 then n0 := n0+1 else n1 := n1+1 fi;
16 od;
17 res := 4*(n00*n00+n01*n01+n10*n10+n11*n11)/(n-1);
18 res := res-2/n*(n0*n0+n1*n1)+1;
19 return (evalf(res))
20 end;
```