Preuve et Analyse des Algorithmes

Examen Module I41, Licence Informatique, USTV.

janvier 2009

↝ *Le sujet est composé trois exercices à traiter en moins de deux heures. Tous les documents sont autorisés. Vous êtes invités à remettre une copie claire, concise, sans rature ni surcharge. Il est par ailleurs inutile de recopier l'énoncé... La note finale tiendra compte de la présentation générale de la copie.*

1 Sauts de Grenouille

Une grenouille doit gravir les n barreaux d'une échelle par sauts successifs sans jamais reculer. En un seul saut, elle peut monter d'un ou deux barreaux mais pas plus. On note G_n, le nombre de toutes les possiblités : $G_1 = 1$, $G_2 = 2$ et $G_3 = 3$.

(a) Déterminer G_4.

(b) Pour $n \geq 2$, formuler G_n en fonction de G_{n-1} et G_{n-2}.

(c) Calculer G_{11}.

(d) Ecrire une fonction récursive pour calculer G_n.

2 Inverse modulaire

Soit p un nombre premier. On appelle *résidu* modulo p un nombre entier positif strictement inférieur à p.

(a) Comment justifier cette terminologie ?

On sait que tous les résidus non-nuls sont inversibles modulo p. L'inverse de x est un résidu y tel que

$$xy \equiv 1 \mod p.$$

i.e. il existe un entier relatif k tel que $xy + kp = 1$..

(b) Utiliser l'algorithme du pgcd étendu pour calculer l'inverse modulaire de 3 modulo 127. Tracer soigneusement les étapes de l'algorithme dans un tableau de valeurs faisant apparaître les restes et quotients successifs.

(c) Quel est le nombre de réduction modulaire dans ce cas particulier?

(d) Donner une expression du nombre de réduction en fonction de p.

Le petit théorème de Fermat affirme que pour tout résidu x,

$$x^p \equiv x \mod p.$$

(e) Déduire du petit théorème de Fermat une expression de l'inverse de x en fonction de x et p quand x n'est pas nul.

(f) Ecrire une fonction pour calculer l'inverse de x modulo p.

(g) Donner une expression du nombre de multiplications, ainsi que du nombre de réductions en fonction de p, et du poids binaire de $p - 2$.

3 Alice

L'objet de la fonction `alice` du code (1) n'est pas immédiat. Elle se rapporte à un des sujets abordés en cours : la détermination des deux plus grandes valeurs dans un tableau donné.

(a) Tracer soigneusement les valeurs des tableaux `num` et `adv` lors de l'exécution de la fonction pour le tableau de taille 12 ci-dessous :

$$[11, 4, 22, 8, 18, 2, 0, 17, 17, 14, 11, 11]$$

(b) Donner l'expression du nombre de comparaisons portant sur des valeurs du tableau `val` (ligne 12) en fonction de n.

(c) Quelle est la valeur retournée par la fonction ?

(d) Modifier le code de la fonction `alice` pour obtenir les deux plus grandes valeurs du tableau `val`. Vous utiliserez un passage de paramètre par adresse pour transmettre la valeur de la seconde plus petite valeur.

```c
int alice(int val[], int n)
{ int i, j, d;
  int adv[ n ], num[ n ];
  for( i = 0; i < n; i++) {
    num[i] = i;
    adv[i] = -1;
  }
  d = 1;
  while ( d < n ){
    i = 0; j = d;
    while ( j < n ) {
      if ( val[ num[i] ] < val[ num[j] ] ){
        adv[ num[i] ] = num[j];
        num[i]        = num[j];
      } else adv[ num[j] ] = num[ i ];
      i += 2*d;
      j += 2*d;
    }
    d *= 2;
  }
  return val[ num[0] ];
}
```

Figure 1: Une fonction mystérieuse ?