INFO0947: Récursivité et Élimination de la Récursivité

Maxime Deravet, s202214

Table des matières

1 Formulation Récursive

$$\sum_{i=0}^{n-1} a_i \times 16^{n-1}$$

a étant le nombre retourné par convert(),
i le rang dans la chaine de caractères,
et n la taille de la chaine de caractères soustrait du rang

2 Spécification

Préconditions

La chaîne de caractères hexa doit être composée de caractères repris dans l'hexadécimal, autrement dit des chiffres compris entre 0 et 9 ou des lettres de A à F.
n doit être égal à la taille de hexa

Postconditions

La fonction hexa_dec_rec() renvoie l'équivalent du nombre hexadécimal entré en base 10

3 Construction Récursive

Programmation défensive :

On vérifie que le tableau n'a pas de valeur négative, et que les caractères rentrés sont bien hexadécimaux.

```
/**
* Précondition : - Hexa doit être une chaine de caractères composée de chiffres
*compris entre 0 et 9 ou des lettres comprises entre A et F
*                 - n doit être égal à la taille de la chaine hexa
* Postcondition : retourne le nombre hexadécimal entré en base 10
*
*/
unsigned int hexa_dec_rec(char *hexa, int n)
```

Extrait de Code 1 – Programmation défensive

Cas récursif :

Il n'y a qu'un cas récursif, si n >0
(L'assertion vérifie la valeur maximale d'un int, car la fonction unsigned int convert renvoie -1)

```
  if (n >0){
    assert(convert(hexa[n-1]) != 4294967295);
    nombre = convert(hexa[n-1]);
    return nombre + 16 * hexa_dec_rec(hexa, n-1);
  }
```

Extrait de Code 2 – Cas récursif

Cas de base :

Simplement si n == 0, alors la fonction renvoie 0

```c
else
  return 0;
```

Extrait de Code 3 – Cas de base

Code complet :

Le unsigned int nombre est juste là pour rendre le code plus lisible, mais n'a pas d'autres utilités.

```c
/**
* Précondition : - Hexa doit être une chaine de caractères composée de chiffres
*compris entre 0 et 9 ou des lettres comprises entre A et F
*                 - n doit être égal à la taille de la chaine hexa
* Postcondition : retourne le nombre hexadécimal entré en base 10
*
*/
unsigned int hexa_dec_rec(char *hexa, int n){
   assert(n>-1);


  unsigned int nombre;


  if (n >0){
    assert(convert(hexa[n-1]) != 4294967295);
    nombre = convert(hexa[n-1]);
    return nombre + 16 * hexa_dec_rec(hexa, n-1);
  }

else
  return 0;
```

Extrait de Code 4 – Cas de base

4 Traces d'Exécution

5 Complexité

Le corps de la fonction, avec découpe en régions, est le suivant :

```
unsigned int hexa_dec_rec(char *hexa, int n){

  unsigned int nombre;


  if (n >0){
    assert(convert(hexa[n-1]) != 4294967295);
    nombre = convert(hexa[n-1]);
    return nombre + 16 * hexa_dec_rec(hexa, n-1);
  }

else
  return 0;
```

o - b

} - T(n)

} - a

On peut faire une telle découpe. En fait, on considère qu'ils vont impliquer autant d'opérations élémentaires :b. Par contre, le temps pris par l'appel récursif est bienT(n).On obtient alors le système suivant :

$$T(n) = \begin{cases} a & \text{si } n = 0, \\ b + T(n) & \text{sinon} \end{cases}$$

Il y a bien une addition entre b et T(n) parce qu'il faut d'abord faire b opérations,puis, il faut exécuter 1 appel récursif. Résolvons l'équation de récurrence par application de la méthode de proche en proche :

$$\begin{aligned} T(n) &= b + T(n) && (1)\\ &= 2 \times b + 2 \times T(n) && (2)\\ &= 3 \times b + 3 \times T(n) && (3)\\ &= \ldots && (4)\\ &= k \times b + k \times T(n) && (5)\\ & && (6) \end{aligned}$$

On sait que :
T(1) =b+T(0) =b+a

Essayons de faire apparaitre T(1) à la place de T(n) :
T(1) = b + T(0) = b + a
n = 1
n = k

Il vient alors :

$$\begin{align}
T(n) &= k \times b + k \times T(n) \tag{7} \\
&= b \times n + b + a \in O(n) \tag{8} \\
\tag{9}
\end{align}$$

Nous avons donc une complexité linéaire

6 Dérécursification