Option Informatique
Complexité

Corrigé

30 septembre 2006

1 Différence symétrique de deux ensembles

Question 1 *Le neutre de Δ est $\emptyset$. Si χ_A désigne la fonction caractéristique de A, alors $\chi_{A\,\Delta\,B} = \chi_A \oplus \chi_B$; l'associativité de $\oplus$ permet de conclure.*

Question 2 *$A\,\Delta\,B$ est contenu dans $A \cup B$, qui est fini. $|A\,\Delta\,B| >= 0$ (banalement), l'égalité étant obtenue ssi $A = B$. $|A\,\Delta\,B| <= |A \cup B| <= |A| + |B|$; on a l'égalité ssi A et B sont disjoints.*

Question 3
$x \in A_1\,\Delta\,A_2\,\Delta \ldots \Delta\,A_n$ ssi $\chi_{A_1\,\Delta\,A_2\,\Delta \ldots \Delta\,A_n}(x) = 1$, soit $\chi_{A_1}(x) \oplus \cdots \oplus \chi_{A_n}(x) = 1$; le calcul se faisant dans $\mathbb{Z}/2\mathbb{Z}$, la somme du membre de gauche vaut 1 ssi elle contient un nombre impair de 1. Conclusion : $A_1\,\Delta\,A_2\,\Delta \ldots \Delta\,A_n$ est l'ensemble des éléments de $A_1 \cup A_2 \cup \ldots \cup A_n$ qui appartiennent à un nombre impair d'ensembles A_i.

Question 4 *On rédige une fonction auxiliaire* `diff` *qui calcule la différence ensembliste $A \setminus B$. On peut alors construire les deux listes qui représentent $A \setminus B$ et $B \setminus A$; comme ces deux ensembles n'ont aucun élément en commun, l'opérateur @ de concaténation de listes suffit pour construire la réunion.*

```
let rec diff b = function
 | [] → []
 | t : :q when mem t b → diff b q
 | t : :q → t : :(diff b q) ; ;
```

```
let diffsym a b = (diff a b) @ (diff b a) ; ;
```

Question 5 *Le calcul de* `mem t b` *requiert q comparaisons ; ce calcul est effectué p fois, donc le calcul de $A \setminus B$ coûte pq comparaisons. Par raison de symétrie, le calcul de $B \setminus A$ a le même coût, si bien que le coût total est $2pq$.*

Question 6 *On compare les têtes de listes : en cas d'égalité, on jette les deux. En cas d'inégalité, on prend la plus petite et on laisse la plus grande en attente. À chaque étape, on extrait au moins un élément de l'une des deux listes, au prix de deux comparaisons au plus. Donc le coût est majoré par $2(p + q)$. Remarquons que la liste construite est classée en ordre décroissant, d'où le* `rev` *final.*

```
let diffsym_2 a b =
 let rec aux accu = function
  | ([],[]) → rev accu
  | ([],t2 : :q2) → aux (t2 : :accu) ([],q2)
  | (t1 : :q1,[]) → aux (t1 : :accu) (q1,[])
  | (t1 : :q1,t2 : :q2) when t1=t2
                        → aux accu (q1,q2)
  | (t1 : :q1,(t2 : :q2 as l2)) when t1<t2
                  → aux (t1 : :accu) (q1,l2)
  | (l1,t2 : :q2) → aux (t2 : :accu) (l1,q2)
 in aux [] (a,b) ; ;
```

2 Calcul efficace de x^n

Question 7 `let rec es x n = if n=1 then x else x * (es x (n-1)) ; ;`

Le nombre de multiplications est égal à $\boxed{n-1.}$

Question 8 *Supposons que* $n = \overline{a_p \dots a_0}^2$ *avec* $a_p = 1$. *Posons* $n_p = 1$ *et* $n_k = \overline{a_p \dots a_k}^2$ *pour* $0 \leq k \leq p-1$.

Au départ, on initialise la variable y avec la valeur x.

Montrons qu'à la fin de chaque boucle portant sur k, la variable y contient x^{n_k}.

A l'entrée de la première boucle, y contient $x^{n_p} = x$.

Supposons qu'à l'entrée de la boucle indexée par k, y contienne $x^{n_{k+1}}$.

Or $n_k = 2\,n_{k+1} + a_k$, *donc* $x^{n_k} = x^{2\,n_{k+1}+a_k} = \left(x^{n_{k+1}}\right)^2 x^{a_k} = y^2\,x^{a_k}$. *Donc pour avoir* x^{n_k}, *il faut bien d'abord élever y au carré (c'est à dire remplacer y par* $y * y$*), puis remplacer y par* $y * x$ *lorsque* $a_k = 1$. *C'est justement ce que propose l'algorithme de Legendre, donc à la sortie de la boucle, y contient bien* x^{n_k}.

Question 9

```
let calcul T x =
let l = vect_length T and res = ref 1 in
for i = 0 to (l-1) do
  res := (!res*!res);
  if T.(i) = 1 then
    res :=!res * x;
done;
!res;;
```

Question 10 $\lambda(n) = p+1$. *La boucle sur k est effectuée p fois (k variant de* $p-1$ *à 0), ce qui donne* $p = \lambda(n)-1$ *élévations au carré. Le nombre de multiplications de y par x est égal au nombre de* a_k *égaux à 1 pour* $0 \leq k \leq p-1$, *c'est donc* $\nu(n)-1$.

Le nombre de multiplications est donc égal à $\boxed{\lambda(n)+\nu(n)-2.}$

3 Calcul de terme de suite

Question 11

```
let u_rec(n) = match n with
| 0 → 1.0
| _ → let s = ref 0.0 in
       for k = 1 to n do
       s :=!s
            +. u_rec(n-k)/.float_of_int(k)
       done;
       !s
;;
```

```
let u_iter(n) =
  let v = make_vect (n+1) 0.0 in
  v.(0) <- 1.0;
  for p = 1 to n do
    for k = 1 to n do
      v.(p) <- v.(p)
               +. v.(p-k)/.float_of_int(k)
    done
  done;
  v.(n)
;;
```

Question 12 *Soient* $\mathcal{T}_{rec}(n), \mathcal{M}_{rec}(n), \mathcal{T}_{iter}(n)$ *et* $\mathcal{M}_{iter}(n)$ *les complexités temporelles et spatiales de* `u_rec` *et* `u_iter` :

On a de manière immédiate $\mathcal{T}_{iter}(n) = O(n^2)$ *et* $\mathcal{M}_{iter}(n) = O(n)$. *Par ailleurs, il existe a et b constantes telles que :*

$$\begin{aligned} \mathcal{T}_{rec}(n) &= a + bn + \Sigma_{k=1}^{n} \mathcal{T}_{rec}(n-k) \\ &= a + bn + \Sigma_{k=0}^{n-1} \mathcal{T}_{rec}(k) \\ &= 2\mathcal{T}_{rec}(n-1) + b \end{aligned}$$

donc $\mathcal{T}_{rec}(n) \sim \lambda 2^n$ *pour un certain* $\lambda > 0$.

On suppose que les variables locales sont libérées dès qu'elles ne sont plus nécessaires. Comme `u_rec` *utilise un nombre constant de mémoires locales, le nombre de positions mémoire nécessaire au calcul de* `u_rec` *est proportionnel au nombre maximal d'appels imbriqués, soit* $\mathcal{M}_{rec}(n) = O(n)$

Question 13

```
let u_rec(n) =
  if n = 0 then 1 else u_rec(n/2) + u_rec(n/3)
;;
```

```
let u_iter(n) =
  let v = make_vect (n+1) 0 in
  v.(0) <- 1;
  for i = 1 to n do v.(i) <- v.(i/2)
                             + v.(i/3)
```

```
  done ;
   v.(n)
 ;;
```

Question 14 *On a de manière immédiate* $\mathcal{T}_{iter}(n) = O(n)$ *et* $\mathcal{M}_{iter}(n) = O(n)$.

Par ailleurs $\mathcal{T}_{rec}$ *vérifie la relation :*

$$\mathcal{T}_{rec}(n) = a + \mathcal{T}_{rec}(\lfloor n/2 \rfloor) + \mathcal{T}_{rec}(\lfloor n/3 \rfloor)$$

Soit $\alpha > 0$. *En posant* $\mathcal{U}(n) = \frac{\mathcal{T}_{rec}(n)+a}{n^\alpha}$ *on obtient :*

$$\mathcal{U}(n) \leq \frac{1}{2^\alpha}\mathcal{U}(\lfloor n/2 \rfloor) + \frac{1}{3^\alpha}\mathcal{U}(\lfloor n/3 \rfloor)$$

2

Donc si l'on a $\frac{1}{2^\alpha} + \frac{1}{3^\alpha} \leq 1$ *alors la fonction* $\mathcal{U}$ *est bornée, et l'on en déduit que* $\mathcal{T}_{rec}(n) = O(n^\alpha)$. *Comme* $\frac{1}{2^\alpha} + \frac{1}{3^\alpha} = 1$ *admet l'unique racine* $\alpha \approx 0.79$, *on a finalement* $\mathcal{T}_{rec}(n) = O(n^{0.79})$. *Enfin,* $\mathcal{M}_{rec}(n)$ *est proportionnel au nombre maximal d'appels à* `u_rec` *imbriqués, soit* $\mathcal{M}_{rec}(n) = O(\ln n)$.

La méthode récursive naïve s'avère donc plus efficace que la méthode récursive avec stockage des résultats intermédiaires !