DM : Multiplication de polynômes et transformée de Fourier

Augustin Lucas MP2I

I Méthode naïve

1. Multiplication naïve :

```
let mul_poly_naive p q =
  let np = Array.length p in
  let nq = Array.length q in
  let n = np + nq -1 in
  let r = Array.make n 0. in
  let tmp = ref 0. in
  for k=0 to n-1 do
    tmp := 0.;
    for i=0 to k do
      if (i < np) && ((k-i) < nq) then
        tmp := !tmp +. p.(i) *. q.(k-i)
      else ();
    done;
    r.(k) <- !tmp;
  done;
  r;;
```

2. La complexité de la fonction précédente est en $O(n^2)$ car deux boucles `for` sont imbriquées

II Nombres complexes

1. Définition de 0 et 1

```
let zero = {
    re=0.;
    im=0.
};;

let un = {
    re=1.;
    im=0.
};;
```

2. Conjugué d'un nombre complexe :

```
let conj z =
  {re = z.re; im = -.z.im};;
```

3. Somme de deux complexes :

```
let add z1 z2 =
  {re = z1.re +. z2.re; im = z1.im +. z2.im};;
```

4. Produit de deux complexes :

```
let mult z1 z2 =
  let re = z1.re*.z2.re-.z1.im*.z2.im in
  let im = z1.re*.z2.im+.z2.re*.z1.im in
  {re = re; im = im};;
```

III Transformée de Fourier

1. Méthode de Horner :

```
let rec horner p x = match p with
  | [] -> zero;
  | e::q -> add e (mult x (horner q x));;
```

L'intérêt de cette méthode de Horner est de pouvoir évaluer un polynôme en utilisant une méthode récursive, ayant une complexité en $O(n)$

2. Fonction divise :

```
let rec divise l = match l with
  | [] -> ([], [])
  | [e] -> ([e], [])
  | e1::e2::q -> let q1, q2 = divise q in
                 (e1::q1, e2::q2);;
```

3. fft :

```
let rec fft p w =
  let q1, q2 = divise p in
  let list = ref (merge (fft q1 (mult w w))
  (List.map (fun x -> mult x w)(fft q2 (mult w w)))) in
    let w_puiss = ref w in
    while !w_puiss.re != 1. do
      if !w_puiss.im > 0. then ()
      else (
        list := (horner p !w_puiss)::!list
      );
      w_puiss := mult !w_puiss w;
    done;
    !list;;
    (* On utilise la récursivité pour calculer les n' premiers éléments
        et la méthode de Horner pour la suite *)
```

4. *On néglige la fonction* `divise`*, de complexité* $O(\frac{n}{2})$ À chaque appel récursif, la méthode de Horner, de complexité $O(n)$ est appelée $\frac{n}{2}$ fois, et `fft` est

appelée $\frac{n}{2}$ fois également. En notant $C(n)$ la complexité d'un appel récursif on a :
$C(n) = C(\frac{n}{2}) + O(\frac{n}{2})$
$C(n) = C(\frac{n}{4}) + O(\frac{n}{4}) + O(\frac{n}{2})$

En notant $n = 2^m$ *(on suppose n une puissance de 2)*, on a donc m appels récursifs, soit $log_2(n)$ donc la fonction `fft` est de complexité $O(log_2(n))$

5. puissance de 2 :

```
let rec est_puiss2 n = match n with
  | 1 -> true
  | _ -> n mod 2 = 0 && est_puiss2 (n/2);;

let rec puiss2 l =
  if est_puiss2 (List.length l) then l
  else puiss2 (l @ [zero]);;
```

IV Multiplication de polynômes

1. Compléter une liste de taille n :

```
let completer l =
    let t = List.length l in
    let rec aux n l1 =
      if n = 0 then l1
      else aux (n-1) (zero::l1) in
    l @ (aux (t-1) []);;
```

2. Multiplication de transformées de Fourier :

```
let mul_ft l p =
  let l' = completer (puiss2 l) in
  let p' = completer (puiss2 p) in
  let n = float (List.length l') in
  let w = exp (mult {im=Float.pi*.2.;re=0.} {re=1./.n;im=0.}) in
  let l_ft = fft l' w in
  let p_ft = fft p' w in
  let rec aux l p = match l, p with
    | [], [] -> []
    | el::l', ep::p' -> (mult el ep)::(aux l' p')
    | _, _ -> failwith "Les polynômes initiaux ne sont pas de la même taille" in
  aux l_ft p_ft;;
```

3. Coefficients de R :

```
let coeff r =
  let n = (List.length r)/2 in
  let rec aux l = match l with
    | [] -> []
```

```
    | e::q -> (mult e {im=0.;re=1./.(float n)})::(aux q) in
  aux r;;
```

4. Multiplication de polynômes :

```
let mul_poly p q =
  let r = mul_ft p q in
  let n =  float (List.length r) in
  let w = exp (mult {im=Float.pi*.2.;re=0.} {re=1./.n;im=0.}) in
  let r' = fft r w in
  coeff r';;
```

5. Calculs de complexité :

- Complexité de `fft` :
  (question 4) : $O(log_2(n))$
- Complexité de `completer` :
  $t-1$ appels de la fonction `aux` et une complexité en $O(t)$ pour le `@`, avec t la taille de la liste en argument. La complexité est donc en $O(2t)$ environ.
- Complexité de `puiss2` :
  Quadratique, en $O(n^2)$ en raison de l'opération `@`
- Complexité de `exp` :
  On suppose la complexité de `exp` en $O(1)$
- Complexité de `mul_ft` :
  En notant n la taille de `puiss2 l`, que l'on suppose égal à `puiss2 p`, et t la taille de l et p, on a :
  $O(\text{mul\_ft}) = 2 * (O(2n) + O(t^2)) + 2 * O(log_2(n))$
  $O(\text{mul\_ft}) = 2 * O(2(n))$
  $O(\text{mul\_ft}) = O(n)$
  `mul_ft` est donc de complexité linéaire
- Complexité de `coeff` :
  Complexité linéaire, en $O(List.length\ r)$
- Complexité de `mul_poly` :
  La complexité de `mul_poly` est donc linéaire, relative aux tailles de p et q, en $O(n)$