Programmation en Caml

Partie I - Parcours en escalier dans un tableau

1) On remplit d'abord la première colonne avec les puissances de 2.
On déduit ensuite chaque colonne de la précédente en la multipliant par 3.

```
let remplir n =
  let m= make_matrix (n+1) (n+1) 1 in
           (* m.(0).(0) est initialise' a` 1 *)

           (* premie`re colonne *)
  for i=1 to n do m.(i).(0) <- 2 * m.(i-1).(0) done;

           (* colonnes suivantes *)
  for j=1 to n do
        for i=0 to n do m.(i).(j) <- 3 * m.(i).(j-1) done done;
  m;;
```

2) Le nombre de multiplications est égal à $(n+1)^2 - 1 = \underline{n^2 + 2\,n}$.

3) On utilise la règle donnant le PGCD lorsque les deux nombres sont factorisés en produit de nombres premiers.

```
let pgcd m i j k l =
   let p = min i k and q = min j l in m.(p).(q);;
```

4) On cherche le plus grand élément inférieur ou égal à p sur chaque ligne (s'il existe).
Sur la ligne d'indice 0, on part de la droite, jusquà trouver un élément $m_{0,j_0} \leq p$.
A l'entrée de la première boucle et à la sortie de chaque boucle indexée par i_0, (i, j) est la position du plus grand des éléments $\leq p$ situés sur les lignes d'indice allant de 0 à i_0 (invariant de boucle).
Quand on passe de (i_0, j_0) à $(i_0 + 1, j_0 - 1)$, il est inutile de de comparer m_{i_0+1,j_0-1} à $m_{i,j}$:
en effet $m_{i_0+1,j_0-1} = \dfrac{2}{3}\, m_{i_0,j_0} \leq \dfrac{2}{3}\, m_{i,j} < m_{i,j}$, ce qui économise un test.
Lorsqu'on est dans la colonne d'indice 0 et que l'étape suivante conduit à $j = -1$, il n'y a alors plus rien faire dans les lignes suivantes.

```
let pgm m p =
      let n =(vect_length m) -1 in
      let i = ref(0) and j = ref(n) and j0 = ref(n) in
      while m.(0).(!j0) > p do j0 := !j0 - 1 done;
      j := !j0;                    (* ligne d'indice 0 *)

      for i0=1 to n do             (* lignes suivantes *)
         if !j0 >= 0  && m.(i0).(!j0) > p
         then  j0 := !j0 -1        (* pas de mise a` jour de i et j *)
         else if !j0 >= 0 &&  m.(!i).(!j) < m.(i0).(!j0) then
             begin i := i0; j := !j0 end
      done;
      (!i, !j,  m.(!i).(!j)) ;;
```

5) Appelons curseur le couple (i_0, j_0) : celui-ci a trois sortes de déplacements possibles :

$\longleftarrow$ sur la première ligne d'indice 0, $\swarrow$ ou $\downarrow$ ensuite.

Le déplacement vertical utilise deux comparaisons, les deux autres une seule.

- S'il n'y a pas de déplacement horizontal sur la première ligne, alors à chaque boucle on descend si possible d'une ligne, ce qui donne au plus $2\,n$ comparaisons (majorant atteint lorsque $p = m_{n,n}$).
- S'il n'y a eu d'abord p déplacements horizontaux sur la première ligne, alors ensuite, chaque déplacement vertical du curseur est automatiquement suivi d'un déplacement oblique car :

  si $m_{i_0,j_0} \leq p$ et $m_{i_0,j_0+1} > p$ et $m_{i_0+1,j_0} \leq p$, alors $m_{i_0+2,j_0} = 4\,m_{i_0,j_0} > 3\,m_{i_0,j_0} = m_{i_0,j_0+1} > p$.

  Il y a donc au plus $n-p$ déplacements obliques et $\left\lceil \frac{n-p}{2} \right\rceil$ déplacements verticaux du curseur, d'où un nombre de comparaisons majoré par : $p + (n-p) + 2\left\lceil \frac{n-p}{2} \right\rceil = n + (n-p+1) \leq 2\,n$ car $p \leq 1$.

Au total, il y a au plus $2\,n$ comparaisons mettant en jeu un élément de la matrice M.

Partie II - Codage de mots en partie commune

```
let creer_branche m =
   let n = string_length m in
   let rec aux i =
   (* i est un indice de position, ce qui evite de couper m *)
         if i = n then noeud(`.`,nil,nil)
         else noeud(m.[i],aux (i+1),nil)
   in aux 0 ;;

let etete m =    (* suppression du premier caracte`re d'une chaine *)
     sub_string  m 1 (string_length m - 1) ;;

let rec placer m a =
   if m="" then a
   else match a with
         |         nil  -> creer_branche m
         | noeud(c,g,d) -> if c = m.[0]
                           then noeud(c,placer (etete m) g, d)
                           else noeud(c, g, placer m d) ;;

let rec inserer m a =
   if m="" then a
   else match a with
       |         nil  -> creer_branche m
       | noeud(c,g,d) -> if c = m.[0]
                         then noeud(c,inserer(etete m) g, d)
                         else if c < m.[0]
                              then noeud(c, g, inserer m d)
                              else noeud(m.[0], creer_branche(etete m), a) ;;
           (* on tient compte cette fois de l'ordre alphabe'tique *)

let affiche a =
    let rec parcours  aa accu = match aa with
         |          nil -> ()
         | noeud(c,g,d) -> if c=`.`

             (* accu contient le mot lu sur le chemin allant
                de la racine au noeud actuellement visit\'e *)

                    then begin
```

```
                                print_string accu;
                                print_newline(); parcours d accu
                              end
                         else begin
                                parcours g (accu^(string_of_char c));
                                parcours d accu
                              end
       in parcours a "";;

    let present m a =
       let rec parcours  aa accu = match aa with
           |          nil -> false
           | noeud(c,g,d) ->
              if c='.'
              then m=accu || parcours d accu
              else parcours g (accu^(string_of_char c)) || parcours d accu
        in parcours a "";;

   let saisir () =
      let mot = ref ("") in
         while !mot <> "fin" do
             mot := read_line();
             if !mot <> "fin" then dico := inserer !mot !dico
         done ;;
```

Partie III - Expressions préfixées bien formées

1) Tout mot bien formé $u = (u_1, \dots, u_n)$ est de longueur impaire car nécessairement $u_n = -1$ et $\sum_{i=1}^{n-1} u_i = 0$, ce qui montre que le préfixe $u' = (u_1, \dots, u_{n-1})$ contient autant de -1 que de 1, donc $n-1$ est pair.

2) Vérifions que w satisfait à la CNS **(1)**.

. La somme des éléments de w est égale à $1 + \sum_{i=1}^{n} u_i + \sum_{j=1}^{m} v_j = 1 - 1 - 1 = -1$.

. Montrons que $\forall k \in [\![1, m+n]\!],\ W_k = \sum_{i=1}^{k} w_i \geq 0$:

- c'est évident si $k = 1$ car $W_1 = w_1 = 1$.
- c'est vrai si $1 \leq k \leq n$ car $W_k = 1 + \sum_{i=1}^{k-1} u_i \geq 1 + 0 = 1$.
- c'est vrai si $k = n+1$ car $W_{n+1} = 1 + \sum_{i=1}^{n} u_i = 1 - 1 = 0$.
- c'est vrai si $n+2 \leq k \leq n+m$ car $W_k = W_{n+1} + \sum_{i=n+2}^{k} w_i = \sum_{j=1}^{k-n-1} v_j \geq 0$ puisque $1 \leq k-n+1 \leq m-1$.

3) Réciproque Puisque w est bien formé de longueur $n > 1$, nécessairement $n \geq 3$, $w_1 = 1$, $w_n = -1$ et $W_{n-1} = 0$.

Forme nécessaire de ℓ (unicité).

L'étude faite au **2)** montre que d'une part $W_{\ell+1} = \sum_{i=1}^{\ell+1} w_i = 0$ et d'autre part que $\forall k \in [\![1; \ell]\!],\ W_{k+1} = \sum_{i=1}^{k+1} w_i \geq 1$.

Ainsi l'indice $p = \ell + 1$ est nécessairement égal à $\operatorname{Min}\{k \in [\![2, n-1]\!] \ / \ W_k = 0\}$.
<u>Synthèse</u> (existence).
Comme $W_{n-1} = 0$, on peut considérer $p = \operatorname{Min}\{k \in [\![2, n-1]\!] \ / \ W_k = 0\}$.
Posons $\ell = p - 1$, $u = (w_2, \ldots, w_p)$ et $v = (w_{p+1}, \ldots, w_n)$. (Remarque : $|u| \geq 1$ et $|v| \geq 1$).

. <u>u est bien formé</u> car $\forall k \in [\![2; p-2]\!]$, $\displaystyle\sum_{i=2}^{k} w_i = W_k - W_1 \geq 1 - 1 = 0$ car $W_k \geq 1$ (puisque $k < p$) et $\displaystyle\sum_{i=2}^{p} w_i = W_p - W_1 \geq 0 - 1 = -1$.

. <u>v est bien formé</u> car $\forall k \in [\![p+1; n-1]\!]$, $\displaystyle\sum_{i=p+1}^{k} w_i = W_k - W_p = W_k \geq 0$ et $\displaystyle\sum_{i=p-1}^{n} w_i = W_n - W_p = -1 + 0 = -1$.

4-5-6-7)

```
let est_bien_forme u =
    let n = u.(0) and i = ref (1) and s = ref (0)  in
    while (!s >=0) && (!i <= n) do
      s := !s + u.(!i);
      i := !i + 1;
    done;
    (!i > n) && (!s = -1) ;;

let construire u v =
    let m =  u.(0) and n = v.(0) in
    let w = make_vect (m+n+2) 1 in
      w.(0) <- m+n+1;
      for i = 1 to m do w.(i+1)   <- u.(i) done;
      for j = 1 to n do w.(j+m+1) <- v.(j) done;
      w;;

let fusion l1 l2 =
    let rec aux  u l1 = match l1 with
       |   []  -> []
       | v::q  -> (construire u v)::(aux u q) in
    let rec parcours liste = match liste with
       |   []  -> []
       | u::q  -> (aux u l2)@(parcours q) in
    parcours l1;;

  let enumere p =
    let t = make_vect (p+1) [] in
    t.(0) <- [[|1;-1|]];
    for k=1 to p do
      for i=0 to k-1 do t.(k) <- t.(k)@fusion (t.(i)) (t.(k-1-i))
      done
    done;
    t;;
```

Notons F l'ensemble des mots bien formés.
La question **2)** a montré que si $u \in F$ et $v \in F$, alors $w = 1uv \in F$.
La question **3)** a montré que si $w \in F$ et $|w| > 1$, alors u se décompose de façon unique en $w = 1uv$ avec $u \in F$ et $v \in F$.
Ainsi on obtient une liste de tous les mots de F de longueur $2k+1$ en réunissant les listes résultat de la "fusion" d'une liste des mots de longeur $2*i+1$ et d'une liste des mots de longeur $2*j+1$ pour tous les couples (i,j) d'entiers ≥ 0 tels que $i+j = k-1$, ce que réalise la fonction `enumere`.

8) Soit $u = (u_1, \dots, u_n)$ de sorte que $\displaystyle\sum_{j=1}^{n} u_j = -1$. Notons $U_k = \displaystyle\sum_{j=1}^{k} u_j$.

<u>Forme nécessaire de i</u> (unicité)

Supposons $(u_{i+1}, \dots, u_n, u_1, \dots, u_i)$ bien formé.

$\forall\, k \in [\![1; i-1]\!],\ 0 \leq \displaystyle\sum_{j=i+1}^{n} u_j + U_k = U_n - U_i + U_k = -1 + U_k - U_i$, donc $U_i < U_k$.

$\forall\, k \in [\![i+1; n]\!],\ 0 \leq \displaystyle\sum_{j=i+1}^{k} u_j = U_k - U_i$, donc $U_i \leq U_k$.

Ainsi $i = \text{Min}\{k \in [\![1, n]\!] \ / \ U_k \text{ est minimum}\}$.

<u>Synthèse</u> (existence)

Posons $i = \text{Min}\{k \in [\![1, n]\!] \ / \ U_k \text{ est minimum}\}$ et vérifions que $v = (u_{i+1}, \dots, u_n, u_1, \dots, u_i)$ est bien formé.

. Si $i + 1 \leq k \leq n$, alors $\displaystyle\sum_{j=i+1}^{k} u_j = U_k - U_i \geq 0$.

. Si $1 \leq k \leq i$, alors $\displaystyle\sum_{j=i+1}^{n} u_j + \sum_{j=1}^{k} u_j = U_n - U_i + U_k = U_k - U_i - 1 \geq 1 - 1 = 0$ car $U_k - U_i \geq 0$ et $U_k \neq U_i$ par définition de i puisque $k < i$.

. $\displaystyle\sum_{j=1}^{n} v_j = \sum_{j=1}^{n} u_j = -1$.

9) Le nombre de façons de placer les k symboles **1** dans une suite de $2k+1$ éléments est égal à C_{2k+1}^{k}.

Parmi les $2k+1$ mots se déduisant par permutation circulaire, il y en a un et un seul qui est bien formé.

Ainsi $\boxed{C_k = \dfrac{C_{2k+1}^{k}}{2k+1}}$

Cela donne notamment $C_1 = 1,\ C_2 = 2,\ C_3 = 5$.

<u>Remarque</u> : C_k est le nombre (dit de Catalan) d'arbres binaires localement complets ayant k noeuds internes et $k+1$ feuilles, ce qui est normal car à tout mot bien formé est associé un tel arbre et réciproquement.

$*$ FIN $*$