OPTION INFORMATIQUE
Devoir Surveillé 1
Corrigé

1 Stratégies d'exponentiation

1. Notons $(n_0, \dots, n_r)$ la suite associée à un calcul de a^n. On montre par récurrence que

$$\forall k \in [|0, r|],\ n_k \leq 2^k$$

   - Initialisation : le résultat est vrai initialement car $n_0 = 1 = 2^0$.
   - Hérédité : supposons le résultat vrai jusqu'à un rang $k \in [|0, r-1|]$. Il existe alors $i, j \leq k$ tels que $n_{k+1} = n_i + n_j$ et par hypothèse de récurrence,

$$n_{k+1} \leq 2^i + 2^j \leq 2^k + 2^k = 2^{k+1}$$

     ce qui prouve le résultat au rang $k+1$.

   On en déduit que $n = n_r \leq 2^r$, c'est à dire que $\log_2(n) \leq r$. r étant entier, on a $\lceil \log_2(n) \rceil \leq r$. Un minorant du nombre de multiplications effectuées est donc $\lceil \log_2(n) \rceil$.
   Si $n = 2^k$ alors, on peut considérer la suite $1, 2, 4, \dots, 2^{k-1}, 2^k$ qui montre que a^n peut être calculé en $k = \log_2(n) = \lceil \log_2(n) \rceil$ multiplications. Ceci donne une infinité de valeurs pour lesquelles le minorant trouvé est optimal.

2. On a les résultats suivants.
   - Pour a^{15} : $(1, 2, 3, 6, 7, 14, 15)$ de longueur 7.
   - Pour a^{16} : $(1, 2, 4, 8, 16)$ de longueur 5.
   - Pour a^{27} : $(1, 2, 3, 6, 12, 13, 26, 27)$ de longueur 8
   - Pour a^{125} : $(1, 2, 3, 6, 7, 14, 15, 30, 31, 62, 124, 125)$ de longueur 12.

3.
```
let par_division n =
    let rec aux n =
        if n=1 then [1]
        else begin
            let l= aux (n/2) in
            let t=hd l in
            if n mod 2 = 0 then (2*t)::l
            else (2*t+1)::(2*t)::l
            end
    in
    inverse (aux n);;
```

4. On montre $\forall n \in \mathbb{N}^*, P_n$: la longueur l_n de `par_division n` vérifie $l_n \leq 2\lfloor \log_2(n) \rfloor + 1$ par récurrence sur n :
   - Si $n = 1$, $l_n = 1 \leq 1$
   - Soit $n > 1$ tel que P_k est vrai jusqu'au rang $n-1$.
     Par hypothèse de récurrence, on a :
     $l_{\lfloor n/2 \rfloor} \leq 2\lfloor \log_2(\lfloor n/2 \rfloor) \rfloor + 1 \leq 2\lfloor \log_2(n/2) \rfloor + 1 = 2\lfloor \log_2(n) - 1 \rfloor + 1 = 2\lfloor \log_2(n) \rfloor - 1$
     Par ailleurs, on a $l_n \leq l_{\lfloor n/2 \rfloor} + 2$, d'où $l_n \leq 2\lfloor \log_2(n) \rfloor + 1$.

   Le cas le pire est celui où l'on tombe toujours sur des exposants impairs. Prenons le cas où $n = 2^k - 1$, qui s'écrit en binaire avec k bits égaux à 1. Notons C_k le nombre de multiplications pour $n = 2^k - 1$. On a $C_1 = 0$ et $C_k = 2 + C_{k-1}$. Ainsi, $C_k = 2(k-1)$ qui est bien égal à $\lfloor \log_2(2^k - 1) \rfloor$.

5. On a les résultats suivants.
   - Pour a^{15} : $(1, 2, 3, 4, 7, 8, 15)$ de longueur 7.
   - Pour a^{16} : $(1, 2, 4, 8, 16)$ de longueur 5.
   - Pour a^{27} : $(1, 2, 3, 4, 8, 11, 16, 27)$ de longueur 8
   - Pour a^{125} : $(1, 2, 4, 5, 8, 13, 16, 29, 32, 61, 64, 125)$ de longueur 12.

6.
```
let rec binaire_inverse n=
    if n=0 then []
    else (n mod 2)::(binaire_inverse (n/2));;
```

7.
```
let par_decomposition_binaire n =
     let rec aux l p s= match l with
     (*p est la puissance de 2 courante et s la dernière somme partielle calculée *)
       |[] -> []
       |0::q -> p::(aux q (2*p) s)
       |1::q -> let s' = p+s in
                p::(s')::(aux q (2*p) (s'))
     in
     let a::q = binaire_inverse n in
     1::(aux q 2 a);;
```

8. On peut utiliser la suite constituée de $1 = 3^0$ et des $2 \times 3^p, 3^{p+1}$ pour $p = 0, \ldots, k-1$. C'est une suite convenable car 2×3^p s'obtient en sommant deux fois le terme qui le précède et 3^{p+1} en sommant les deux termes qui le précèdent. On obtient une suite de longueur $2k+1$. Dans le cas de 27, c'est la suite
$$(1, 2, 3, 6, 9, 18, 27)$$
On obtient une suite de longueur 7, meilleure que celle de l'algorithme *par_division*.

9.
```
let suite_3 n =
     let rec aux n =
         if n=1 then [1]
         else n::(2*(n/3))::(aux (n/3))
     in inverse (aux n);;
```

10. On remarque cette fois que $5^k = 2 \times 2 \times 5^{k-1} + 5^{k-1}$. On peut alors utiliser la suite constituée de 1 et des $2 \times 5^p, 2 \times 2 \times 5^p, 5^{p+1}$ pour $p = 0, \ldots, k-1$. Cette suite est de longueur $3k+1$. Pour $n = 125$, on obtient la suite
$$(1, 2, 4, 5, 10, 20, 25, 50, 100, 125)$$
On obtient une suite de longueur 10 meilleure que celle obtenue avec *par_division*.

11. La suite $(1, 2, 3, 5, 10, 15)$ convient. Les algorithmes précédents ne sont donc pas optimaux (ce que montre aussi la question 20 ou la question 22).

12. Pour $i = k-1$, on cherche un $j \leq i$ tel que $t.(j) + t.(i) = t.(k)$. Si on n'en trouve pas, on cherche avec $k-2$ etc. En faisant les choses dans cet ordre, on obtiendra le plus grand des couples (s'il en existe au moins un).
A i fixé, si on ne trouve pas de j convenable, on finit par obtenir $j = -1$ et il faut alors poursuivre la recherche. Ceci explique la valeur initiale donnée à j.

```
let chercher_indice t k =
   let j=ref (-1) and i=ref (k-1) in
   while !i>=0 && !j=(-1) do
      j:= !i;
      while !j>=0 && t.(!j)+t.(!i)<>t.(k) do decr j done;
      if !j=(-1) then decr i
      done;
   (!j,!i);;
```

La complexité de cette fonction est $O(k^2)$.

13. On utilise un tableau `tab` de flottants de même taille que l'argument `t`. L'idée est de remplir ce tableau afin que `tab.(i)` contienne finalement `x` à la puissance `t.(i)`. Pour cela, on peut utiliser un remplissage de gauche à droite d'après la propriété des suites.

```
let puissance x t =
   let n=Array.length t in
   let tab=Array.make n x in
   for k=1 to (n - 1) do
      let (i,j)=chercher_indice t k in
      tab.(k) <- tab.(i) *. tab.(j)
      done;
   tab.(n-1);;
```

14. On considère un algorithme récursif auxiliaire qui prend en argument :
    - l'entier n pour lequel on cherche une suite optimale ;
    - une liste d'éléments *pris* avec les éléments déjà mis dans la suite ;
    - la liste non vide *aprendre* des éléments encore possibles à ajouter, c'est-à-dire inférieurs à n et correspondant à la somme de deux éléments de *pris*.

    Le but de la fonction est de renvoyer une complétion optimale de *pris*. Il y a deux façons de compléter :
    - en utilisant le premier élément x de *aprendre* et il suffit alors de faire un appel récursif avec $x :: pris$ et les possibles éléments que l'on pourra prendre ensuite (ce sont ceux somme de deux éléments de $x :: pris$, qui sont $> x$ et $\leq n$). Cet appel récursif ne doit être fait que si $x \neq n$ pour respecter les préconditions. Dans le cas où $x = n$, la meilleure complétion est $x :: pris$.
    - sans utiliser cet élément et il s'agit alors de faire un appel récursif avec *pris* et le reste des éléments de *aprendre*. Cet appel ne doit être fait que si *aprendre* a au moins deux éléments pour respecter la précondition.

    Il suffit alors d'appeler cette fonction avec $pris = [1]$ et $aprendre = [2]$.

15.
```
let rec somme y x l n = match l with
   |[] -> []
   |z::q -> if (y+z)<=n && (y+z)>x then (y+z)::(somme y x q n) else (somme y x q n);;


let rec union l1 l2 =
        match l1,l2 with
            [],_ -> l2
            |_,[] -> l1
            |a1::q1 , a2::q2 -> if a1 = a2 then union q1 l2
                                else if a2 > a1 then a2::(union q2 l1)
                                else a1::(union q1 l2);;


let rec suivants l x n =
    match l with
   |[] -> []
   |y::q -> union (somme y x l n) (suivants q x n);;

let rec suite_optimale_rec n pris possible=
     match possible with
     |[x] -> if x=n then n::pris
             else suite_optimale_rec n (x::pris) (suivants (x::pris) x n)
     |x::q -> if x=n then n::pris
```

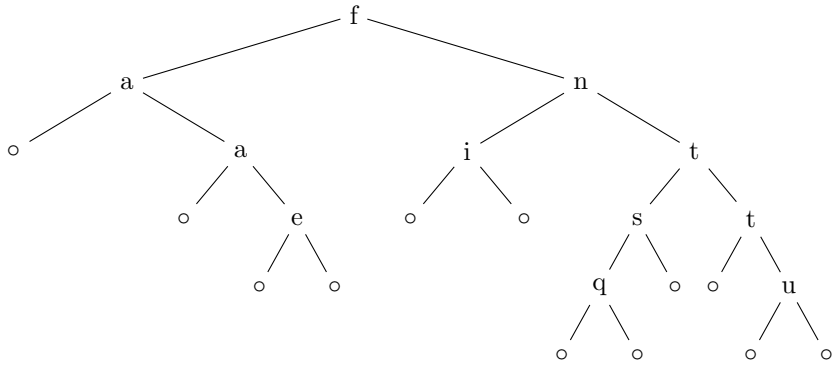
```
                else begin
                   let l1=suite_optimale_rec n (x::pris) (suivants (x::pris) x n)
                   and l2=suite_optimale_rec n pris q
                   in if (longueur l1)<=(longueur l2) then l1 else l2
                   end;;

let suite_optimale n = inverse (suite_optimale_rec n [] [1]);;
```

2 Étude de classes sylvestres

2.1 Algorithme d'insertion dans un arbre sylvestre

16. L'ABR associé au mot "fantastique" :



17.
```
let rec insertion_lettre a t = match t with
  | Vide -> Noeud(Vide,a,Vide)
  | Noeud(tg,r,td) when r <= a -> Noeud(tg,r,insertion_lettre a td)
  | Noeud(tg,r,td) -> Noeud(insertion_lettre a tg,r,td);;
```

18.
```
let rec insertion_mot w t = match w with
  | [] -> t
  | a::q -> insertion_mot q (insertion_lettre a t);;
```

19. On a :

$$\begin{aligned} W_T &= sW_gW_d \\ &= s(yW_gW_d)(eW_gW_d) \\ &= sy(l\varepsilon\varepsilon)(v\varepsilon\varepsilon)e(sW_g\varepsilon)(\varepsilon rW_d) \\ &= sylve(st\varepsilon\varepsilon)r(e\varepsilon\varepsilon) \\ &= sylvestre \end{aligned}$$

20.
```
let rec prefixe t = match t with
  | Vide -> []
  | Noeud(tg,r,td) -> r::(prefixe tg)@(prefixe td);;
```

21. On propose de démontrer le résultat par induction structurelle.

- Si $T = \circ$ alors $w_T = \varepsilon$ et $\circ$ est associé à ε.
- On suppose que $T = (T_g, r, T_d)$ et que la proposition est vraie pour les arbres T_g et T_d. Alors $w_T = rw_gw_d$ où w_g et w_d sont les lectures préfixes de T_g et T_d respectivement. Par hypothèse d'induction T_g et T_d sont respectivement les arbres associés à w_g et w_d respectivement.
  Puisque T est un ABR toutes les lettres de w_g sont strictement inférieures à r et toutes les lettres de w_d sont supérieures ou égales à r, ce qui permet de conclure que T est l'ABR associé à w_T.

Le résultat est donc démontré par induction structurelle.

2.2 Une relation d'équivalence sur les mots

22. On notera $\sim$ la relation binaire sur Σ^* " être S-équivalent à ".

- **Réflexivité :**
  En considérant $n = 0$, on a bien que pour tout $u \in \Sigma^*$, $u \sim u$.
- **Symétrie :**
  On considère u et v dans Σ^* tels que $u \sim v$. Si $n = 0$ le résultat est immédiat, sinon on pose :
  $$\forall i \in [\![0, n-1]\!], \quad w'_i = w_{n-i}$$
  Puisque la relation de S-adjacence est symétrique, pour tout $i \in [\![0, n-1]\!]$ w'_i est S-adjacent à w'_{i+1}, donc $w'_0 \sim w'_n$ ce qui revient à $v \sim u$.
- **Transitivité :**
  On considère u, v et x dans Σ^* tels que $u \sim v$ et $v \sim x$.
  On peut considérer deux nombres entiers naturels n et p, $(w_0, \dots, w_n) \in (\Sigma^*)^{n+1}$, $(y_0, \dots, y_p) \in (\Sigma^*)^{p+1}$ tels que :
  — $u = w_0$, $v = w_n$, pour tout $i \in [\![0, n-1]\!]$, w_i est S-adjacent à w_{i+1}.
  — $v = y_0$, $x = y_p$, pour tout $i \in [\![0, p-1]\!]$, y_i est S-adjacent à y_{i+1}.
  On pose alors pour $k \in [\![0, n+p]\!]$:
  $$z_k = \begin{cases} w_k & \text{si } k \in [\![0, n]\!] \\ y_{k-n} & \text{si } k \in [\![n+1, n+p]\!] \end{cases}$$
  Puisque $w_n = y_0$, par construction on a pour tout $i \in [\![0, n+p-1]\!]$, z_i S-adjacent à z_{i+1}, ce qui permet de conclure que $u \sim x$.

23. Si T contient la lettre b, il admet au moins un noeud.
Pour $T = (\circ, b, \circ)$, on a :
$$T \leftarrow ac = ((\circ, a, \circ), b, (\circ, c, \circ)), \qquad T \leftarrow ca = ((\circ, a, \circ), b, (\circ, c, \circ))$$
le résultat est donc vérifié pour les arbres à 1 noeud.
On considère $n \in \mathbb{N}^*$ et on suppose le résultat vrai pour les arbre contenant b et comptant k noeuds pour tout $k \in [\![1, n]\!]$.
On considère T un arbre à $n+1$ noeuds contenant b. On pose $T = (T_g, r, T_d)$.

- Si $r = b$, on a :
  $$T \leftarrow ac = (T_g \leftarrow a, b, T_d) \leftarrow c = (T_g \leftarrow a, b, T_d \leftarrow c)$$
  De même :
  $$T \leftarrow ca = (T_g \leftarrow a, b, T_d \leftarrow c)$$
- Si $r < b$ et $a < r$ on a :
  $$T \leftarrow ca = T \leftarrow ac = (T_g \leftarrow a, r, T_d \leftarrow c)$$
- Si $r < b$ et $a \geq r$ on a :
  $$T \leftarrow ac = (T_g, r, T_d \leftarrow ac), \qquad T \leftarrow ca = (T_g, r, T_d \leftarrow ca)$$
  Mais dans ce cas T_d est un arbre contenant nécessairement b et comptant moins de n noeuds. On peut lui appliquer l'hypothèse de récurrence pour conclure que $T \leftarrow ac = T \leftarrow ca$.
- Si $r > b$, on reprend le principe précédent appliqué à T_g pour démontrer que $T \leftarrow ac = T \leftarrow ca$.

On a donc démontré le résultat souhaité par récurrence sur le nombre de noeuds de T.

24. Il suffit de démontrer le résultat pour les mots S-adjacents.
On considère donc u et v deux mots S-adjacents, on reprend les notations de l'énoncé :

$$u = \mathtt{f_1bf_2acf_3}, \quad v = \mathtt{f_1bf_2caf_3}$$

On a :

$$\begin{aligned}(\circ \leftarrow u) &= (((\circ \leftarrow \mathtt{f_1bf_2}) \leftarrow \mathtt{ac}) \leftarrow \mathtt{f_3}) \\ (\circ \leftarrow v) &= (((\circ \leftarrow \mathtt{f_1bf_2}) \leftarrow \mathtt{ca}) \leftarrow \mathtt{f_3})\end{aligned}$$

Or l'arbre $(\circ \leftarrow \mathtt{f_1bf_2})$ contient la lettre b. Le résultat de la question précédente assure alors que :

$$((\circ \leftarrow \mathtt{f_1bf_2}) \leftarrow \mathtt{ac}) = ((\circ \leftarrow \mathtt{f_1bf_2}) \leftarrow \mathtt{ca})$$

Ce qui permet de conclure.

25. (a) Si u et v sont S-adjacents alors $|u| = |v|$, deux mots S-équivalents ont donc aussi la même longueur. En particulier :

$$A \subset \Sigma^{|w|}$$

Puisque $\Sigma^{|w|}$ est fini, A est fini.

(b) Deux mots S-adjacents commencent par la même lettre, donc deux mots S-équivalents commencent par la même lettre. Tous les mots de A commencent donc par la lettre r.

(c) $N(a)$ est un ensemble fini et non-vide. On définit alors j_0 par :

$$j_0 = min(j \in [\![1, n-1]\!], \quad (i, j) \in N(a))$$

et i_0 par :

$$i_0 = \max(i \in [\![1, n-1]\!], \quad (i, j_0) \in N(a))$$

Montrons par l'absurde que $i_0 + 1 = j_0$.
Si ce n'était pas le cas, $i_0 < i_0 + 1 < j_0$ et :

$$a_{i_0+1} < r \quad \vee \quad a_{i_0+1} \geq r$$

Dans le premier cas $(i_0 + 1, j_0) \in N(a)$ ce qui contredit la définition de i_0.
Dans le second cas $(i_0, i_0 + 1) \in N(a)$ ce qui contredit la définition de j_0.
Finalement $j_0 = i_0 + 1$ et en posant $k = i_0$ on a $(k, k + 1) \in N(a)$.

(d) Si $N(a) = \emptyset$. Dans le cas où $n = 1$ le résultat est immédiat en posant $u = \varepsilon$ et $v = \varepsilon$.
Si $n > 1$, on peut considérer x mot non vide tel que $a = rx$.
L'un au moins des deux ensembles suivants est alors non-vide :

$$\{i \in [\![1, n]\!], \quad a_i \geq r\}, \quad \{i \in [\![1, n]\!], \quad a_i < r\}$$

Si le premier ensemble est non-vide, on peut poser :

$$k = \min(i \in [\![1, n]\!], \quad a_i > r)$$

Alors puisque $N(a) = \emptyset$ on a pour tout $j \in [\![1, n]\!]$:

$$(j > k) \Rightarrow (a_j \geq r)$$

$u = a_1 \dots a_{k-1}$ et $v = a_k \dots a_n$ conviennent.
Si le premier ensemble est vide, on pose $k = \max(i \in [\![1, n]\!], \quad a_k < r)$ et de la même manière $u = a_1 \dots a_k$ et $v = a_{k+1} \dots a_n$ conviennent.

(e) On considère donc $a \in A$ tel que $(N(a))$ soit minimal. Si $N(a) \neq \emptyset$ on peut considérer $k \in [\![1, n-2]\!]$ tel que $(k, k+1) \in N(a)$, le mot a' obtenu à partir de a en échangeant les lettres a_k et a_{k+1} est alors S-adjacent à a et donc élément de A, mais $(N(a')) < (N(a))$ ce qui contredit la définition de a.
Nécessairement $N(a) = \emptyset$ ce qui permet de conclure grâce au résultat de (d).

26. Si $|w| = 0$ le résultat est immédiat.
Hérédité :
On considère $n \in \mathbb{N}$ et on suppose le résultat vrai pour les mots w tels que $|w| \leq n$.
On considère w un mot de longueur $n+1$. En notant r la première lettre de w, il existe deux mots u et v vérifiant les conditions de la question 25 tels que w soit S-équivalent à ruv.
Le résultat de la question 24 assure que $(\circ \leftarrow w) = (\circ \leftarrow ruv)$.
Par construction :
$$w_{(\circ \leftarrow ruv)} = r w_{(\circ \leftarrow u)} w_{(\circ \leftarrow v)}$$
Puisque $|u| \leq n$ et $|v| \leq n$ l'hypothèse de récurrence assure que $u \sim w_{(\circ \leftarrow u)}$ et $v \sim w_{(\circ \leftarrow v)}$.
On conclut en remarquant que si $u_1 \sim v_1$ et $u_2 \sim v_2$ alors $u_1 u_2 \sim v_1 v_2$. En effet, si $u \sim v$ alors pour tout mot x on a :
$$ux \sim vx \qquad xu \sim xv$$
On a donc :
$$\underbrace{r w_{(\circ \leftarrow u)} w_{(\circ \leftarrow v)}}_{= w_{(\circ \leftarrow ruv)}} \sim ruv \sim w$$
l'égalité $(\circ \leftarrow w) = (\circ \leftarrow ruv)$ permet de conclure.

27. Le résultat de la question **??** assure que si $u \sim v$ alors u et v sont dans la même classe sylvestre.
Le résultat de la question **??** établit la réciproque.

2.3 Construction des classes sylvestres

28. On a :
$$abba \bowtie ba = \{baabba, bababa, babbaa, ababba, abbaba, abbbaa, abbaab\}$$

29.
```
let rec ajout_lettre (a:char) liste = match liste with
  | [] -> []
  | w::lw -> (a::w)::(ajout_lettre a lw);;
```

30.
```
let rec shuffle u v = match (u,v) with
  | [],v -> [v]
  | u,[] -> [u]
  | a::up,b::vp ->
   (ajout_lettre a (shuffle up (b::vp) ))
   @
   (ajout_lettre b (shuffle (a::up) vp ));;
```

31. On vide les deux listes définissant les deux langages jusqu'à obtenir les listes vides :

```
let rec shuffle_l l m = match l,m with
  | [],_ -> []
  | _,[] -> []
  | [u], v::lv -> (shuffle u v) @ (shuffle_l [u] lv)
  | u::lu, [v] -> (shuffle u v) @ (shuffle_l lu [v])
  | u::lu, v::lv ->
   ((shuffle_l [u] (v::lv))
   @
   (shuffle_l (u::lu) [v])) @ (shuffle_l lu lv);;
```

32. On s'appuie sur la propriété admise :

```
let rec sylvestre_T t = match t with
  | Vide -> [[]]
  | Noeud(tg,r,td) -> ajout_lettre r (shuffle_l (sylvestre_T tg) (sylvestre_T td));;
```