ECOLE POLYTECHNIQUE
ECOLES NORMALES SUPERIEURES

CONCOURS D'ADMISSION 2021

MARDI 13 AVRIL 2021
14h00 - 18h00

FILIERE MP - Epreuve n° 4

INFORMATIQUE A (XULCR)

Durée : 4 heures

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve

Cette composition ne concerne qu'une partie des candidats de la filière MP, les autres candidats effectuant simultanément la composition de Physique et Sciences de l'Ingénieur.
Pour la filière MP, il y a donc deux enveloppes de Sujets pour cette séance.

Facteurs dans les mots binaires

Ce sujet traite de mots sur un alphabet $\{0,1\}$. On s'intéresse à leurs facteurs, notamment aux facteurs répétés dans un mot et aux mots ayant le maximum de facteurs distincts. L'algorithmique des facteurs est particulièrement utile dans le domaine de la bio-informatique, où l'analyse de mots très longs, à savoir les génomes, requiert des structures de données efficaces.

Mots. Un *mot* m est une suite finie $m_0 m_1 \dots m_{\ell-1}$ de lettres de l'alphabet $\{0,1\}$. Sa *longueur* ℓ est notée $\ell(m)$ et sa lettre d'indice i est notée m_i pour $0 \leq i \leq \ell(m)-1$. Le *mot vide*, de longueur 0, est noté ε. On note c^ℓ le mot de longueur ℓ formé de ℓ caractères c.

Pour deux mots $m = m_0 m_1 \dots m_{\ell(m)-1}$ et $m' = m'_0 m'_1 \dots m'_{\ell(m')-1}$, on définit leur *concaténation* comme le mot $mm' = m_0 m_1 \dots m_{\ell(m)-1} m'_0 m'_1 \dots m'_{\ell(m')-1}$ de longueur $\ell(mm') = \ell(m) + \ell(m')$.

Pour $0 \leq i \leq j \leq \ell(m)$, on note $m[i..j[$ le mot $m_i m_{i+1} \dots m_{j-1}$, qui est appelé *facteur* de m. On note que le mot vide est un facteur de n'importe quel mot, obtenu pour $0 \leq i = j \leq \ell(m)$. Un *préfixe* (resp. *suffixe*) de m est un facteur de la forme $m[0..j[$ (resp. $m[i..\ell(m)[$) et on le note $m[..j[$ (resp. $m[i..[$). On note

- $\mathrm{F}(m) = \{m[i..j[\mid 0 \leq i \leq j \leq \ell(m)\}$ l'ensemble des facteurs de m ;
- $\mathrm{P}(m) = \{m[..j[\mid 0 \leq j \leq \ell(m)\}$ l'ensemble des préfixes de m ;
- $\mathrm{S}(m) = \{m[i..[\mid 0 \leq i \leq \ell(m)\}$ l'ensemble des suffixes de m.

Arbres binaires. Un *arbre binaire* est défini récursivement de la manière suivante :

- soit comme l'arbre vide, noté V, qui ne contient aucun nœud ;
- soit comme un *nœud*, noté $N(x,g,d)$ et appelé *racine*, où x est l'information stockée dans le nœud, g est un arbre binaire appelé *sous-arbre gauche* et d est un arbre binaire appelé *sous-arbre droit*.

On note $n(a)$ le nombre de nœuds et $h(a)$ la hauteur d'un arbre binaire a, définis par

$$n(V) = h(V) = 0, \quad n(N(x,g,d)) = 1 + n(g) + n(d) \quad \text{et} \quad h(N(x,g,d)) = 1 + \max(h(g), h(d)).$$

Langage OCaml. Ce sujet utilise les listes et les tableaux d'OCaml. Une liste est construite à partir de la liste vide `[]` et de la construction `x :: l` qui renvoie une nouvelle liste dont la tête est l'élément `x` et dont la queue est la liste `l`. L'appel de `List.rev l` renvoie une nouvelle liste, formée des éléments de la liste `l` en ordre inverse.

On peut créer des tableaux avec les deux fonctions `Array.make` et `Array.of_list`. L'appel de `Array.make n x` crée un tableau de taille `n` dont toutes les cases contiennent la valeur `x`. L'appel de `Array.of_list l` crée un tableau contenant, dans l'ordre, les éléments d'une liste `l`. Les cases d'un tableau sont numérotées à partir de 0. La fonction `Array.length` renvoie la taille d'un tableau. Pour un tableau `tab`, on accède à l'élément d'indice `i` avec `tab.(i)` et on le modifie avec `tab.(i) <- v`.

Dans tout le sujet, on représente un mot par un tableau d'entiers, ne contenant que des entiers 0 ou 1. On se donne donc le type suivant :

```
type mot = int array (* ne contient que des 0/1 *)
```

Dans les fonctions demandées, on ne cherchera jamais à vérifier que les tableaux passés en arguments ne contiennent que des 0 et des 1.

Complexité. Par *complexité* (en temps) d'un algorithme A on entend le nombre d'opérations élémentaires (comparaison, addition, soustraction, multiplication, division, affectation, test, etc.) nécessaires à l'exécution de A dans le cas le pire. Lorsque la complexité dépend d'un ou plusieurs paramètres $\kappa_0, \cdots, \kappa_{r-1}$, on dit que A a une complexité en $\mathcal{O}(f(\kappa_0, \cdots, \kappa_{r-1}))$ s'il existe une constante $C > 0$ telle que, pour toutes les valeurs de $\kappa_0, \cdots, \kappa_{r-1}$ suffisamment grandes (c'est-à-dire plus grandes qu'un certain seuil), pour toute instance du problème de paramètres $\kappa_0, \cdots, \kappa_{r-1}$, la complexité est au plus $C \cdot f(\kappa_0, \cdots, \kappa_{r-1})$.

Dépendances. Ce sujet est conçu pour être traité linéairement. La partie I est nécessaire pour la partie II, et ces deux premières parties motivent la partie III, elle-même utilisée dans la question 22 de la partie IV. En revanche, les questions 20 et 21 de la partie IV ainsi que la partie V sont indépendantes du reste du sujet.

Partie I. Arbres de mots

Dans cette partie, on introduit une structure de données, appelée *arbre de mots*, qui représente un ensemble fini de mots. Un arbre de mots est un arbre binaire dont chaque nœud contient un booléen. Un arbre de mots a représente un ensemble fini de mots, noté $\mathbb{M}(a)$, défini comme suit. Si l'arbre a est vide, alors $\mathbb{M}(a)$ est l'ensemble vide. Si l'arbre a est de la forme $N(b, a_0, a_1)$, alors $\mathbb{M}(a)$ est l'ensemble des mots suivants :

- le mot vide ε si le booléen b est `true` ;
- les mots de la forme $0m$ pour $m \in \mathbb{M}(a_0)$;
- les mots de la forme $1m$ pour $m \in \mathbb{M}(a_1)$.

Autrement dit, $\mathbb{M}(a)$ est l'ensemble des mots correspondant aux chemins menant de la racine à un nœud contenant `true`, en utilisant le caractère `0` quand on va à gauche et le caractère `1` quand

on va à droite. Ainsi, l'arbre a_1 de la figure 1 représente l'ensemble de mots $\mathbb{M}(a_1) = \{\varepsilon, 10, 11\}$. Dans tous les dessins, on utilise `T` (resp. `F`) pour le booléen `true` (resp. `false`).

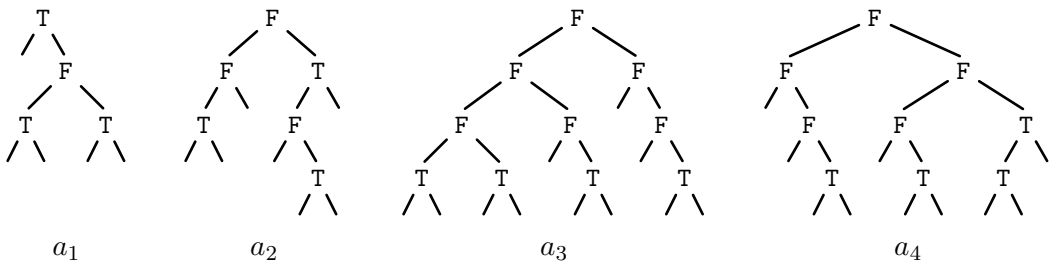

FIGURE 1 – Des arbres de mots.

On note qu'un ensemble de mots peut être représenté par une infinité d'arbres de mots. En effet, il suffit de remplacer un arbre vide par un nœud $N(\texttt{false}, V, V)$ pour obtenir un autre arbre de mots représentant le même ensemble de mots. Par exemple, l'arbre vide V et l'arbre à un nœud $N(\texttt{false}, V, V)$ représentent tous les deux l'ensemble vide. On dit qu'un arbre de mots est *réduit* dès lors que tout nœud dont les deux sous-arbres sont vides contient le booléen `true`. Dans toute la suite, on ne manipulera que des arbres de mots réduits.

Question 1. Donner l'ensemble de mots représenté par l'arbre a_2 de la figure 1. Dessiner un arbre de mots réduit représentant l'ensemble de mots $\{\varepsilon, 00, 010, 1, 110, 111\}$.

Question 2. Montrer que, pour tout ensemble fini de mots, il existe un unique arbre de mots réduit qui le représente.

Dans la suite, on note $\mathbb{A}(M)$ l'arbre de mots réduit de l'ensemble fini de mots M. On a donc $\mathbb{M}(\mathbb{A}(M)) = M$ pour tout ensemble fini de mots M et inversement $\mathbb{A}(\mathbb{M}(a)) = a$ pour tout arbre de mots réduit a.

Pour représenter un arbre de mots en OCaml, on se donne le type suivant

```
type am =
  | V
  | N of bool * am * am
```

où `V` représente l'arbre vide et `N` représente un nœud. Par exemple, l'arbre a_1 de la figure 1 est représenté par la valeur suivante :

```
N (true, V, N (false, N (true, V, V), N (true, V, V)))
```

Question 3. Écrire une fonction `zeros_puis_uns: int -> am` qui reçoit en argument un entier $n \geq 0$ et qui renvoie l'arbre de mots réduit représentant tous les mots de la forme 0^p1^q avec $p + q = n$. L'arbre a_3 de la figure 1 donne un exemple d'un tel arbre pour $n = 3$.

Question 4. Écrire une fonction `k_uns: int -> int -> am` qui reçoit en arguments deux entiers $k \geq 0$ et $n \geq 0$ et qui renvoie l'arbre de mots réduit représentant tous les mots de

longueur au plus n et contenant exactement k caractères 1. L'arbre a_4 de la figure 1 donne un exemple d'un tel arbre pour $k = 2$ et $n = 3$.

Indication : Pour s'assurer que l'arbre renvoyé est bien réduit, on pourra se servir de la fonction suivante.

```
let sN = function
  | (false, V, V) -> V
  | (b, a0, a1) -> N (b, a0, a1)
```

Question 5. Écrire une fonction `compter: am -> int` qui reçoit en argument un arbre de mots a et renvoie le cardinal de l'ensemble $\mathbb{M}(a)$. La complexité en temps doit être linéaire en $n(a)$, mais il n'est pas demandé de la justifier.

Question 6. Écrire une fonction `chercher: am -> mot -> bool` qui reçoit en arguments un arbre de mots a et un mot m et détermine si m appartient à l'ensemble $\mathbb{M}(a)$. La complexité en temps doit être linéaire en $\ell(m)$, mais il n'est pas demandé de la justifier.

Question 7. Écrire une fonction `enumerer: am -> mot list` qui reçoit en argument un arbre de mots réduit a et renvoie l'ensemble de mots $\mathbb{M}(a)$ sous la forme d'une liste. La complexité en temps doit être linéaire en la taille du résultat $\sum_{m \in \mathbb{M}(a)} \ell(m)$. On justifiera la complexité.

Indication : On pourra écrire une fonction récursive auxiliaire qui reçoit en arguments

- *une liste* `acc` *dans laquelle on accumule les mots déjà construits,*
- *une liste* `pref` *contenant les lettres rencontrées le long du chemin menant du nœud courant à la racine,*
- *le sous-arbre* `t` *dont la racine est le nœud courant,*

et qui renvoie la liste `acc` *complétée avec tous les mots de* $\mathbb{M}($`t`$)$ *auxquels on a ajouté le préfixe donné par la liste* `pref`*.*

Question 8. Écrire une fonction `ajouter: am -> mot -> am` qui reçoit en arguments un arbre de mots réduit a et un mot m et renvoie l'arbre de mots réduit représentant l'ensemble de mots $\mathbb{M}(a) \cup \{m\}$. La complexité en temps doit être linéaire en $\ell(m)$, mais il n'est pas demandé de la justifier.

Pour la suite, on remarque que les nœuds de l'arbre de mots $\mathbb{A}(M)$ correspondent à l'ensemble de mots $\{m[..i[\mid m \in M \text{ et } 0 \leq i \leq \ell(m)\}$. En effet, les mots donnés par les chemins de la racine aux nœuds de $\mathbb{A}(M)$ (en utilisant `0` quand on va à gauche et `1` quand on va à droite) sont précisément les préfixes d'au moins un mot de M.

Partie II. Arbre des suffixes

Dans cette partie, on considère un mot m et on définit l'*arbre des suffixes de* m, noté $\mathbb{AS}(m)$, comme l'arbre de mots réduit de tous les suffixes de m, c'est-à-dire $\mathbb{AS}(m) = \mathbb{A}(\mathrm{S}(m))$.

Question 9. Donner l'arbre $\mathbb{AS}(m)$ pour le mot $m = 1011$.

Question 10. Quelle est la hauteur de l'arbre $\mathbb{AS}(m)$? En déduire une fonction `retrouver_mot: am -> mot` qui retrouve le mot m à partir de l'arbre $\mathbb{AS}(m)$ de ses suffixes. La complexité en temps doit être linéaire en $n(\mathbb{AS}(m))$, mais il n'est pas demandé de la justifier.

Question 11. Montrer que les nœuds de l'arbre $\mathbb{AS}(m)$ correspondent aux facteurs distincts de m. En déduire que le nombre de nœuds de l'arbre $\mathbb{AS}(m)$ est au moins linéaire et au plus quadratique en $\ell(m)$. Montrer que ces bornes sont atteintes, c'est-à-dire que pour tout entier ℓ, il existe un mot m de longueur ℓ tel que l'arbre $\mathbb{AS}(m)$ a $\ell+1$ nœuds (resp. au moins $C\ell^2$ nœuds où C est une constante indépendante de ℓ que l'on explicitera).

Des bornes plus précises sur le nombre de facteurs distincts de m seront étudiées dans la partie IV.

Partie III. Arbre des facteurs

Dans cette partie, on introduit une structure de données potentiellement plus compacte pour représenter l'arbre des suffixes d'un mot m fixé, appelée *arbre des facteurs*. Cette structure est basée sur les deux idées suivantes :

- dans l'arbre de suffixes $\mathbb{AS}(m)$, un chemin formé de nœuds dont le booléen est `false` et dont l'un des sous-arbres est vide peut être compacté dans le nœud suivant, quitte à se rappeler de la séquence de caractères correspondante (`0` quand on va à gauche et `1` quand on va à droite) ;
- une telle séquence de caractères se représente de façon compacte par une paire d'entiers (p, q) qui désigne le facteur $m[p..q[$.

On considère donc un arbre binaire dont chaque nœud contient deux entiers p et q tels que $0 \leq p \leq q \leq \ell(m)$ et un booléen. Un tel arbre a représente l'ensemble des mots $\mathbb{M}(a)$ défini comme suit. Si l'arbre a est vide, alors $\mathbb{M}(a)$ est l'ensemble vide. Si l'arbre a est de la forme $N(p, q, b, a_0, a_1)$, alors $\mathbb{M}(a)$ est l'ensemble des mots suivants :

- le mot $m[p..q[$ si le booléen b est `true` ;
- les mots de la forme $m[p..q[0f$ pour $f \in \mathbb{M}(a_0)$;
- les mots de la forme $m[p..q[1f$ pour $f \in \mathbb{M}(a_1)$.

L'arbre a est un arbre des facteurs du mot m si l'ensemble $\mathbb{M}(a)$ est exactement l'ensemble de tous les suffixes de m. On dit qu'un arbre des facteurs est *réduit* dès lors que tout nœud dont au moins un des sous-arbres est vide contient le booléen `true`. Ainsi, l'arbre de la figure 2 est un arbre des facteurs réduit du mot 10110100. Dans tous les dessins, on représente l'information contenue dans un nœud $N(p, q, b, a_0, a_1)$ par $p..q$(`T`) si b est `true` et $p..q$(`F`) si b est `false`.

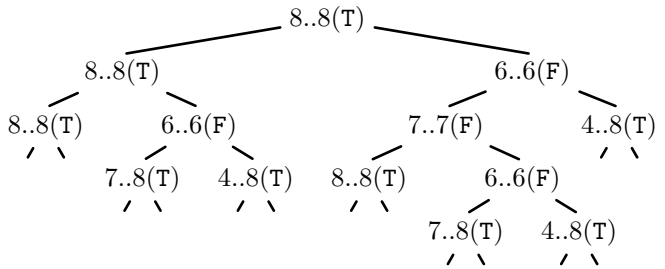

FIGURE 2 – Un arbre des facteurs réduit du mot 10110100.

*Attention : Un arbre des facteurs du mot m n'est pas un arbre de mots de tous les facteurs de m, mais une représentation compacte de l'arbre des suffixes de m. On parle d'*arbre des facteurs *car il permet de manipuler facilement tous les facteurs de m, comme on le verra plus loin.*

Question 12. Pour chaque arbre a de la figure 3, donner l'ensemble $\mathbb{M}(a)$ et indiquer s'il s'agit d'un arbre des facteurs pour le mot $m = 01001$, et s'il est réduit.

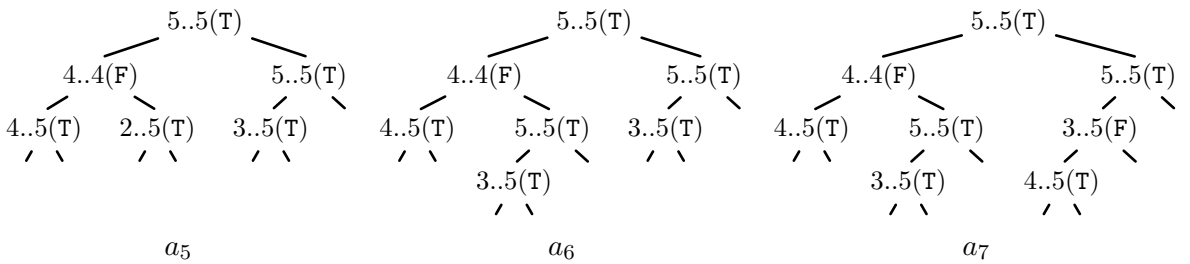

FIGURE 3 – Des arbres.

On peut se convaincre facilement qu'il existe toujours au moins un arbre des facteurs pour tout mot m. En effet, il suffit de construire l'arbre des suffixes $\mathbb{AS}(m)$ de m de la partie II et d'ajouter dans chaque nœud la paire d'entiers $(0, 0)$. Pour obtenir un arbre des facteurs réduit, il suffit alors de réduire cet arbre comme expliqué au début de cette partie, en compactant tous les nœuds dont le booléen est `false` et dont un des sous-arbres est vide. On verra plus loin une procédure pour construire directement un arbre des facteurs réduit.

On montre maintenant qu'un arbre des facteurs réduit du mot m est compact en comparaison de l'arbre des suffixes de la partie II qui pouvait être quadratique comme observé à la question 11.

Question 13. Soit a un arbre des facteurs réduit d'un mot m. On note que l'arbre a a $\ell + 1$ nœuds si m est 0^ℓ ou 1^ℓ pour $\ell \geq 0$. En supposant que les deux lettres 0 et 1 apparaissent dans le mot m,

1. montrer que le nombre de nœuds de a ayant au moins un sous-arbre vide est au plus $\ell(m)$,
2. en déduire une borne supérieure du nombre de nœuds de a en fonction de $\ell(m)$, et montrer que cette borne est atteinte (c'est-à-dire que pour tout $\ell > 0$, il existe un mot de longueur ℓ dont un arbre des facteurs a précisément ce nombre de nœuds).

Pour représenter un arbre des facteurs en OCaml, on se donne le type

```
type af =
  | V
  | N of int * int * bool * af * af
```

où `V` représente l'arbre vide et `N` représente un nœud. On rappelle que le mot m est fixé dans cette partie. On pourra supposer qu'il est dans une variable globale `m`.

Question 14. Écrire une fonction `nb_facteurs: af -> int` qui reçoit en argument un arbre des facteurs a du mot m et renvoie le nombre de facteurs distincts de m. La complexité en temps doit être linéaire en $n(a)$, mais il n'est pas demandé de la justifier.

Question 15. Écrire une fonction `plpc: mot -> int -> int -> mot -> int -> int -> int` (pour "plus long préfixe commun") qui prend en arguments un mot m_1 avec deux indices $0 \leq p \leq q \leq \ell(m_1)$ et un mot m_2 avec deux indices $0 \leq i \leq j \leq \ell(m_2)$, et qui renvoie le plus grand k tel que $p + k \leq q$ et $i + k \leq j$ et $m_1[p..p + k[= m_2[i..i + k[$. La complexité en temps doit être linéaire en le résultat, mais il n'est pas demandé de la justifier.

Question 16. Écrire une fonction `facteur: af -> mot -> bool` qui reçoit en arguments un arbre des facteurs a du mot m et un mot f et détermine si f est un facteur de m. La complexité en temps doit être linéaire en $\ell(f)$. On justifiera la complexité.

Question 17. Écrire une fonction `facteurs: af -> mot list` qui reçoit en argument un arbre des facteurs réduit a du mot m et renvoie l'ensemble des facteurs $\mathrm{F}(m)$ sous la forme d'une liste. Chaque facteur doit apparaître une fois et une seule dans cette liste. La complexité en temps doit être linéaire en la taille du résultat $\sum_{f \in \mathrm{F}(m)} \ell(f)$. On justifiera la complexité.

On montre maintenant comment construire un arbre des facteurs réduit pour le mot m. Pour cela, on part de l'arbre vide et on ajoute successivement tous les suffixes du mot m, dans un ordre arbitraire. Un suffixe $m[i..[$ est ajouté à l'arbre courant a de la manière suivante :

- si $a = V$, on construit l'arbre $N(i, \ell(m), \texttt{true}, V, V)$,
- si $a = N(p, q, b, a_0, a_1)$, on cherche le plus grand k tel que $p + k \leq q$ et $i + k \leq \ell(m)$ et tel que $m[p..p + k[= m[i..i + k[$. On distingue alors deux cas :
  - si $p + k < q$ alors on construit l'arbre $N(p, p + k, b', a'_0, a'_1)$ où
    - si $i+k < \ell(m)$, alors b' est `false` et a'_0 et a'_1 sont les arbres $N(p+k+1, q, b, a_0, a_1)$ et $N(i + k + 1, \ell(m), \texttt{true}, V, V)$, placés dans le bon ordre ;
    - si $i+k = \ell(m)$, alors b' est `true` et a'_0 et a'_1 sont les arbres $N(p+k+1, q, b, a_0, a_1)$ et V, placés dans le bon ordre ;
  - si $p + k = q$, alors
    - si $i + k < \ell(m)$, alors on ajoute récursivement le suffixe $m[i + k + 1..[$ dans le bon sous-arbre de a ;
    - si $i + k = \ell(m)$, alors on remplace le booléen b par `true`.

Dans la suite, on admet que cette procédure construit un arbre des facteurs réduit pour le mot m. On a représenté à la figure 4 les étapes successives de la construction de l'arbre des facteurs pour le mot $m = 1010$, en insérant les suffixes dans deux ordres différents. On peut remarquer que la forme de l'arbre des facteurs construit par cette procédure ne dépend pas de l'ordre d'insertion (en revanche, les indices dans les nœuds dépendent de l'ordre d'insertion), mais cette propriété ne sera pas utilisée par la suite.

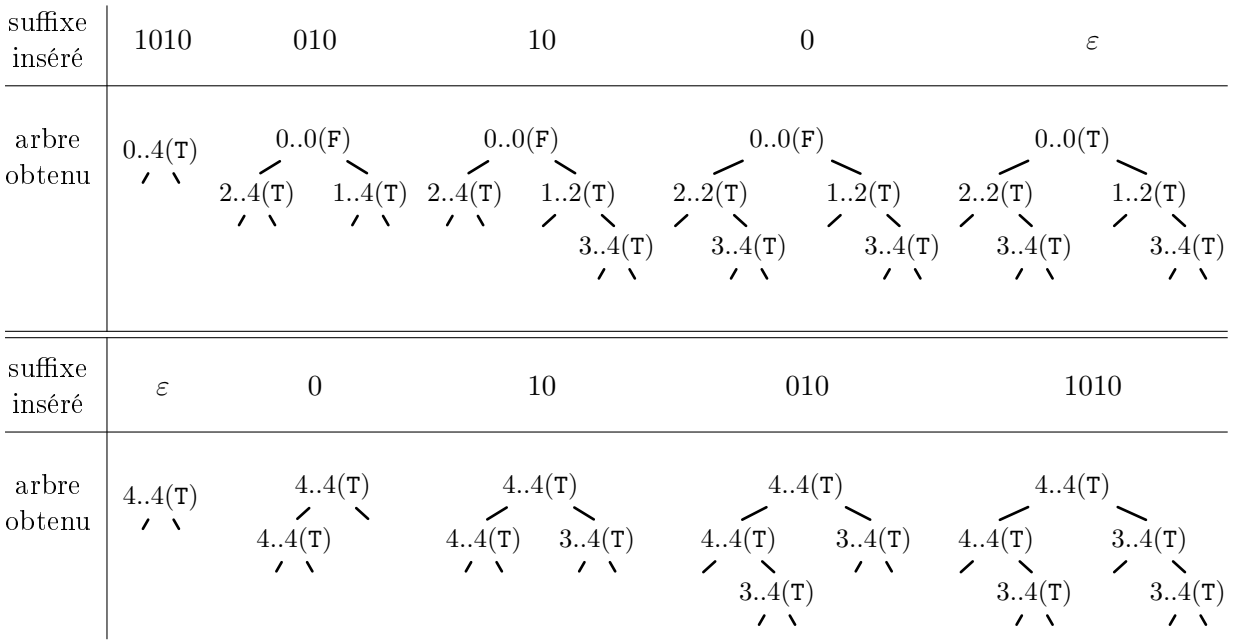

FIGURE 4 – Étapes successives de la construction de l'arbre des facteurs pour le mot $m = 1010$, en insérant les suffixes dans deux ordres différents.

Question 18. Représenter les étapes successives de la construction de l'arbre des facteurs pour le mot $m = 01001$, en insérant successivement les suffixes dans l'ordre ε, 1, 01, 001, 1001, et 01001.

Question 19. Compléter le code de la fonction `ajouter_suffixe: int -> af -> af` ci-dessous, qui reçoit en arguments une position $0 \leq i \leq \ell(m)$ et un arbre a en cours de construction et renvoie l'arbre obtenu en ajoutant le suffixe $m[i..[$ à l'arbre a en suivant la procédure décrite ci-dessus.

```
let rec ajouter_suffixe i a =
  let n = Array.length m in
  match a with
  | V -> N (i, n, true, V, V)
  | N (p, q, b, a0, a1) ->
      let k = plpc m p q m i n in
      if p + k < q then
        (* CAS 1 *)
        ...
      else
        (* CAS 2 *)
        ...
```

On donnera uniquement dans la copie le code correspondant à `(* CAS 1 *)` et `(* CAS 2 *)`. La fonction `plpc` est celle décrite à la question 15.

Partie IV. Application : plus long facteur répété

Dans cette partie, on s'intéresse aux *facteurs répétés* du mot m, c'est-à-dire aux triplets (i, j, k) avec $0 \leq i \leq i + k \leq \ell(m)$ et $0 \leq j \leq j + k \leq \ell(m)$ tels que $i \neq j$ et $m[i..i + k[= m[j..j + k[$. On cherche la longueur maximale d'un facteur répété dans m. On commence par un algorithme naïf.

Question 20. Écrire une fonction `plfr1: mot -> int` (pour "plus long facteur répété") qui reçoit en argument un mot m non vide et renvoie la longueur d'un plus long facteur répété dans m. La complexité en temps doit être $O(\ell(m)^3)$, mais il n'est pas demandé de la justifier.

Pour améliorer la complexité de ce calcul, on utilise maintenant un algorithme de programmation dynamique. Pour $0 \leq i, j \leq \ell(m)$, on note $c(i, j)$ la longueur du plus long suffixe commun de $m[0..i[$ et $m[0..j[$. On rappelle qu'en OCaml, une matrice est un tableau de tableaux. L'appel de `Array.make_matrix p q x` crée une matrice de taille `p` × `q` dont toutes les cases contiennent la valeur `x`. Pour une matrice `mat`, on accède à l'élément (`i`, `j`) avec `mat.(i).(j)` et on le modifie avec `mat.(i).(j) <- v`.

Question 21. Écrire une fonction `plfr2: mot -> int` qui reçoit en argument un mot m non vide et renvoie la longueur d'un plus long facteur répété dans m en construisant la table des $c(i, j)$. La complexité en temps doit être $O(\ell(m)^2)$, mais il n'est pas demandé de la justifier.

On suppose maintenant qu'on connaît un arbre des facteurs a du mot m. On observe que les plus longs facteurs répétés dans le mot m correspondent aux nœuds internes de profondeur maximale de l'arbre des facteurs. La profondeur est ici considérée comme étant égale à la longueur du mot représenté par le chemin depuis la racine, c'est-à-dire que chaque nœud $N(p, q, b, a_0, a_1)$ contribue $q - p + 1$ à la profondeur.

Question 22. Écrire une fonction `plfr3: af -> int` qui reçoit en argument un arbre des facteurs réduit a d'un mot m non vide et renvoie la longueur d'un plus long facteur répété dans m. La complexité en temps doit être en $O(\ell(m))$. On justifiera la complexité.

Partie V. Mot contenant un maximum de facteurs distincts

Dans cette partie, on cherche des mots ayant un maximum de facteurs distincts. Pour un mot m et un entier $0 \leq k \leq \ell(m)$, on note $f_k(m)$ le nombre de facteurs distincts de m de longueur k, et $f(m) = \sum_{k=0}^{\ell(m)} f_k(m)$ le nombre de facteurs distincts de m.

Question 23. Montrer que pour tout mot m sur l'alphabet $\{0, 1\}$ et tout entier $0 \leq k \leq \ell(m)$, on a

$$f_k(m) \leq \min(2^k, \ell(m) - k + 1).$$

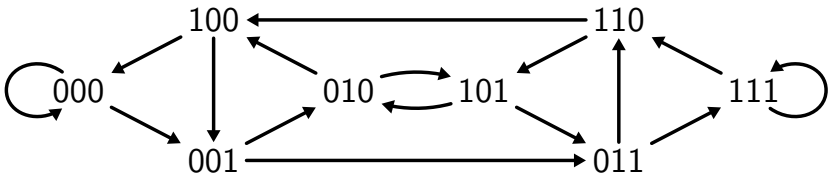

FIGURE 5 – Le graphe G_4.

On considère maintenant un entier k fixé. On suppose qu'il existe un mot m tel que chaque mot de longueur k sur l'alphabet $\{0,1\}$ apparaisse exactement une fois comme facteur de m. Par exemple, le mot 0001011100 a cette propriété pour $k = 3$. On montrera plus tard qu'il tel mot existe toujours.

Question 24. Déterminer la longueur d'un tel mot et que ce mot atteint le nombre maximum de facteurs distincts parmi les mots de longueur $\ell(m)$.

Il reste donc à montrer qu'il existe un mot m tel que chaque mot de longueur k apparaisse exactement une fois comme facteur de m. Pour cela, on va considérer des cycles eulériens dans un graphe orienté particulier, appelé *graphe de de Bruijn*.

Un graphe orienté G est dit *fortement connexe* si, pour toute paire de sommets v et w, il existe un chemin de v à w.

Un graphe orienté G est dit *eulérien* si, pour tout sommet v, le degré entrant de v est égal au degré sortant de v. Un *cycle eulérien* dans un graphe orienté G est un cycle passant une et une seule fois par chaque arête de G.

Question 25. Montrer que tout graphe eulérien G ayant au moins une arête admet un cycle C, et que le graphe G privé des arêtes du cycle C est encore eulérien.

Question 26. En déduire qu'un graphe orienté fortement connexe est eulérien si et seulement s'il contient un cycle eulérien.

Finalement, on considère le graphe orienté G_k dont

- les sommets sont tous les mots à $k-1$ lettres sur l'alphabet $\{0,1\}$,
- les arêtes sont les couples $(m_1 \dots m_{k-1}, m_2 \dots m_k)$ pour tous les mots $m_1 \dots m_k$ à k lettres sur l'alphabet $\{0,1\}$.

Par exemple, le graphe orienté G_4 est représenté à la figure 5.

Question 27. Montrer que le graphe G_k admet un cycle eulérien. En déduire qu'il existe un mot m tel que chaque mot de longueur k apparaisse exactement une fois comme facteur de m.

* *
*