Concours e3a-Polytech, France

Session 2020, filière MP

Informatique

Durée : 4 heures

Ce que traite le sujet

- Partie I : logique et calcul des propositions
- Partie II : le problème de Freudenthal
- Partie III.1 : mots de Lyndon
- Partie III.2 : mots de de Bruijn

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/e3a/mp/2020/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

CONCOURS COMMUN INP

ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE MP

INFORMATIQUE

Jeudi 7 mai : 8 h - 12 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- *Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.*
- *Ne pas utiliser de correcteur.*
- *Écrire le mot FIN à la fin de votre composition.*

Les calculatrices sont interdites

Le sujet est composé de trois parties, toutes indépendantes.

Partie I - Logique et calcul des propositions

Dans la suite, les variables propositionnelles seront notées x_1, x_2.... Les connecteurs propositionnels $\wedge$ (conjonction), $\vee$ (disjonction), $\Rightarrow$ (implication) et $\Leftrightarrow$ (équivalence) seront classiquement utilisés.
De même, la négation d'une variable propositionnelle x_i (respectivement d'une formule $\mathcal{F}$) sera notée $\neg x_i$ (resp. $\neg\mathcal{F}$).

I.1 - Définitions

Définition 1 (Minterme, maxterme)**.**
Soit $(x_1 \cdots x_n)$ un ensemble de n variables propositionnelles.

- *On appelle minterme toute formule de la forme $y_1 \wedge y_2 \wedge \cdots y_n$ où pour tout $i \in \{1, \cdots, n\}$ y_i est un élément de $\{x_i, \neg x_i\}$.*
- *On appelle maxterme toute formule de la forme $y_1 \vee y_2 \vee \cdots y_n$ où pour tout $i \in \{1, \cdots, n\}$ y_i est un élément de $\{x_i, \neg x_i\}$.*

Les mintermes (respectivement maxtermes) $y_1 \wedge y_2 \wedge \cdots y_n$ et $y'_1 \wedge y'_2 \wedge \cdots y'_n$ (resp $y_1 \vee y_2 \vee \cdots y_n$ et $y'_1 \vee y'_2 \vee \cdots y'_n$) sont considérés identiques si les ensembles $\{y_i, 1 \leq i \leq n\}$ et $\{y'_i, 1 \leq i \leq n\}$ le sont.

Q1. Donner l'ensemble des mintermes et des maxtermes sur l'ensemble (x_1, x_2).

Définition 2 (Formes normales conjonctives et disjonctives)**.**
Soit $\mathcal{F}$ une formule propositionnelle qui s'écrit à l'aide de n variables propositionnelles $(x_1 \cdots x_n)$.

- *On appelle forme normale conjonctive de $\mathcal{F}$ toute conjonction de maxtermes logiquement équivalente à $\mathcal{F}$.*
- *On appelle forme normale disjonctive de $\mathcal{F}$ toute disjonction de mintermes logiquement équivalente à $\mathcal{F}$.*

Définition 3 (Système complet)**.**
Un ensemble de connecteurs logiques C est un système complet si toute formule propositionnelle est équivalente à une formule n'utilisant que les connecteurs de C.

Par définition, $C = \{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$ est un système complet.

I.2 - Le connecteur de Sheffer

On définit le connecteur de Sheffer, ou d'incompatibilité, par $x_1 \diamond x_2 = \neg x_1 \vee \neg x_2$.

Q2. Construire la table de vérité du connecteur de Sheffer.

Q3. Exprimer ce connecteur en fonction de $\neg$ et $\wedge$.

Q4. Vérifier que $\neg x_1 = x_1 \diamond x_1$.

Q5. En déduire une expression des connecteurs $\wedge, \vee$ et $\Rightarrow$ en fonction du connecteur de Sheffer. Justifier en utilisant des équivalences avec les formules propositionnelles classiques.

Q6. Donner une forme normale conjonctive de la formule $x_1 \diamond x_2$.

Q7. Donner de même une forme normale disjonctive de la formule $x_1 \diamond x_2$.

Q8. Démontrer par induction sur les formules propositionnelles que l'ensemble de connecteurs $C = \{\diamond\}$ est un système complet.

Q9. Application : soit $\mathcal{F}$ la formule propositionnelle $x_1 \vee (\neg x_2 \wedge x_3)$. Donner une forme logiquement équivalente de $\mathcal{F}$ utilisant uniquement le connecteur de Sheffer.

Partie II - Le problème de Freudenthal (Informatique pour tous)

L'objectif de cette partie est de proposer une implémentation en langage Python d'une solution au problème de Freudenthal.
Hans Freudenthal (1905-1990), mathématicien allemand naturalisé néerlandais, spécialiste de topologie algébrique, est connu pour ses contributions à l'enseignement des mathématiques. En 1969, il soumet à une revue mathématique le problème suivant :

Un professeur dit à ses deux étudiants Sophie et Pierre : "J'ai choisi deux entiers x et y, tels que $1 < x < y$ et $x + y \leq n$. J'ai confié à Pierre la valeur Π du produit de x et y. J'ai confié à Sophie la valeur Σ de la somme de x et y. Pierre, Sophie, je vous demande de trouver x et y."
Pierre et Sophie engagent alors le dialogue suivant :

- *Pierre : "Je ne connais pas les nombres x et y."*
- *Sophie : "Avant même que tu me le dises, je savais déjà que tu ne connaissais pas x et y."*
- *Pierre : "Ah ! eh bien maintenant je connais x et y."*
- *Sophie : "Très bien, mais moi aussi alors maintenant je connais x et y."*

Dans la suite, on note $\mathcal{N}_n = \left\{(x, y) \in \mathbb{N}^2, 1 < x < y \text{ et } x + y \leq n\right\}$.
Si la discussion entre Sophie et Pierre semble stérile, une quantité importante d'informations est cependant échangée qui amène au bout du dialogue à la solution.

Q10. À quelle condition sur x et y Pierre aurait-il pu dire dès le début : "Je connais x et y" ?

Puisque Pierre ne peut répondre tout de suite, cela signifie que le produit Π peut s'écrire pour plusieurs couples d'entiers $(x, y) \in \mathcal{N}_n$.

Q11. Écrire une fonction `CoupleProd(n)` qui renvoie la liste des entiers P pour lesquels il existe au moins deux couples $(x, y) \in \mathcal{N}_n$ tels que $xy = P$. Par exemple, `CoupleProd(9)`=[12] puisque $12 = 3 \times 4 = 2 \times 6$ et qu'aucune autre valeur ne satisfait la propriété.

Sophie savait déjà que Pierre ne connaissait pas la réponse. C'est donc que, pour tout $(x, y) \in \mathcal{N}_n$ qui satisfait $x + y = \Sigma$, le produit xy est dans la liste précédente.

Q12. Soit un entier $S \leq n$. Écrire une fonction `Prod(S,n)` qui renvoie, pour l'ensemble des $(x, y) \in \mathcal{N}_n$ tels que $x + y = S$, la liste des entiers $P = xy$. Par exemple, `Prod(8,9)` retourne [12,15] puisque $8 = 6 + 2 = 3 + 5$.

Q13. Pour $S \leq n$, en déduire une fonction `Candidat_S(n)` qui renvoie la liste des entiers S tels que la liste `Prod(S,n)` est incluse dans la liste `CoupleProd(n)`.

Pierre peut maintenant déduire la valeur de Σ du fait qu'elle appartient à la liste retournée par la fonction `Candidat_S(n)`. Plus précisément, le produit Π n'apparaît dans la liste `Prod(S,n)` que pour une seule valeur de S de la liste `Candidat_S(n)`. Pour déterminer cet unique S, on recherche tout d'abord les produits P pour lesquels :

- il existe deux sommes S_1 et S_2 dans la liste `Candidat_S(n)` telles que $S_1 < S_2$;
- P apparaît dans les listes `Prod(`S_1`,n)` et `Prod(`S_2`,n)`.

Q14. Écrire une fonction `Double_P(n)` qui renvoie la liste des produits P satisfaisant ces deux conditions.

Il reste à construire une fonction `Reste_S(n)` permettant de ne retenir que les sommes S de la liste `Candidat_S(n)` pour lesquelles il existe un unique élément en commun entre les listes `Prod(S,n)` et `Double_P(n)`.

Q15. Écrire une fonction `Reste_S(n)` qui renvoie la liste de ces sommes.

Pour que Pierre conclue, il faut que la liste `Reste_S(n)` soit réduite à un singleton. Pour que Sophie conclue également, il lui suffit de rechercher les éléments de la liste `Prod(S,n)` qui ne sont pas dans `Double_P(n)`.

Q16. Pour $S \leq n$, écrire une fonction `Reste_P(S,n)` qui renvoie la liste de ces produits.

Les deux étudiants connaissent maintenant Σ et Π.

Q17. Pour $S \leq n$, écrire une fonction `Solution(P,S,n)` qui retourne le couple (x, y) recherché.

Partie III - Mots de Lyndon et de de Bruijn

Cette partie comporte des questions nécessitant un code `Caml`. Pour ces questions, les réponses ne feront pas appel aux fonctionnalités impératives de langage (en particulier pas de boucles, pas de références).

On considère ici un alphabet totalement ordonné de k symboles, noté Σ.

III.1 - Mots de Lyndon

III.1.1 - Définitions

Définition 4 (Mot)**.**
Un mot est une suite finie de longueur n de symboles $\mathbf{m} = m_0 \cdots m_{n-1}$ *où pour tout i,* $m_i \in \Sigma$ *et* $n \geq 1$ *est la longueur de* $\mathbf{m}$*. On notera* $n = |\mathbf{m}|$.

On note Σ^n l'ensemble des mots de longueur n construits sur Σ et Σ^* l'ensemble des mots de longueur quelconque construits sur Σ. On note enfin ϵ le mot vide.

Le type `Caml` choisi pour représenter un mot est une chaîne de caractères (`string`). Les éléments de Σ sont représentés par le type `char`.
Dans toute la suite, les seules fonctions/méthodes `Caml` sur les chaînes de caractères qui peuvent être utilisées sont :

- $S.[i]$ (valeur du i^e caractère)
- l'opérateur de concaténation `^`
- la fonction `String.length : string -> int`
  `String.length s` retourne la longueur de `s`.
- la fonction `String.sub : string -> int -> int -> string`.
  `String.sub s start l` retourne une chaîne de longueur `l` contenant la sous-chaîne de `s` qui commence en `start`.

Définition 5 (Préfixe, suffixe)**.**
Soient $\mathbf{m} = m_0 \cdots m_{|m|-1}$ *et* $\mathbf{p} = p_0 \cdots p_{|p|-1}$ *deux mots de* Σ^* *:*

- $\mathbf{mp} = m_0 \cdots m_{|m|-1} p_0 \cdots p_{|p|-1} \in \Sigma^{|m|+|p|}$ *est le concaténé de* $\mathbf{m}$ *et* $\mathbf{p}$.
- $\mathbf{m}$ *est un préfixe de* $\mathbf{p}$ *si* $|m| < |p|$ *et* $m_i = p_i$ *pour* $0 \leq i \leq |m| - 1$.
- $\mathbf{m}$ *est un suffixe de* $\mathbf{p}$ *si* $|m| < |p|$ *et* $m_i = p_{|p|-|m|+i}$ *pour* $0 \leq i \leq |m| - 1$.

On définit alors la relation $\prec$ par :

$$\mathbf{m} \prec \mathbf{p} \Leftrightarrow \quad (\mathbf{m} \text{ est un préfixe de } \mathbf{p}) \text{ ou}$$
$$(\text{Il existe } k \in [\![0, |m| - 1]\!] \text{ tel que pour tout } i \in [\![0, k-1]\!], m_i = p_i \text{ et } m_k < p_k)\,.$$

Q18. On donne le type
`type comparaison = Inferieur | Egal | Superieur ;;`
Écrire une fonction recursive `ordre : string -> string -> comparaison` telle que `ordre m p` est l'ordre relatif des mots `m` et `p`.

Définition 6 (Ordre lexicographique)**.**
La relation définie par : $\mathbf{m} \preceq \mathbf{p}$ *si et seulement si* $\mathbf{m} = \mathbf{p}$ *ou* $\mathbf{m} \prec \mathbf{p}$ *est appelée ordre lexicographique sur* Σ^*.

Définition 7 (Conjugué)**.**
Soit $\mathbf{m} \in \Sigma^n$. *Un conjugué de* $\mathbf{m}$ *est un mot de la forme* $m_i \cdots m_{n-1} m_0 \cdots m_{i-1}$, *pour* $i \in [\![0, n-1]\!]$. *Par convention, le conjugué de* $\mathbf{m}$ *pour* $i = 0$ *est le mot* $\mathbf{m}$ *lui-même.*

Q19. Écrire une fonction Caml `conjugue : string -> int -> string` telle que `conjugue m i` retourne le conjugué de $\mathbf{m}$ débutant par le i^e caractère de $\mathbf{m}$, $0 \leq i \leq |\mathbf{m}| - 1$.

La notion de conjugaison induit une relation C définie sur Σ^* par $\mathbf{m}\, C\, \mathbf{p}$ si et seulement si $\mathbf{p}$ est un conjugué de $\mathbf{m}$.

Q20. Montrer que C est une relation d'équivalence.

Définition 8 (Collier)**.**
Un collier est le plus petit mot dans l'ordre lexicographique d'une classe de mots équivalents par la relation C.
Un collier d'ordre n est dit périodique s'il peut s'écrire $\mathbf{m}^l$, *où* $\mathbf{m} \in \Sigma^r, r \geq 2$ *et* $l > 1$. *Il est dit apériodique sinon, autrement dit si deux conjugaisons non triviales des membres de sa classe d'équivalence ne sont jamais égales.*

Définition 9 (Mot de Lyndon)**.**
Un mot $\mathbf{m} \in \Sigma^*$ *est un mot de Lyndon si c'est un collier apériodique.*

Q21. On suppose 0<1. Pour les mots suivants, indiquer si ce sont ou non des mots de Lyndon. Dans le cas négatif, justifier votre réponse :

(i). 0010011.

(ii). 010011.

(iii). 001001.

Q22. Écrire une fonction Caml `Lyndon : string -> bool` telle que `Lyndon m` renvoie `true` si $\mathbf{m}$ est un mot de Lyndon. Cette fonction fera appel à une fonction récursive.

III.1.2 - Génération de mots de Lyndon

Soit $\mathbf{m} \in \Sigma^*$ un mot de Lyndon. Pour générer à partir de $\mathbf{m}$ un mot de Lyndon $\mathbf{q}$ de longueur au plus $n \geq |m|$ sur Σ, on utilise l'**algorithme 1**.

Algorithme 1 : Algorithme de génération d'un mot de Lyndon

Données : $\mathbf{m} \in \Sigma^*$ un mot de Lyndon, $n \geq |m|$
Résultat : $\mathbf{q}$ le mot de Lyndon généré à partir de $\mathbf{m}$.
**** Etape 1 ****
Concaténer le mot $\mathbf{m}$ à lui même jusqu'à obtenir un mot $\mathbf{q}$ de longueur n. La dernière occurrence de $\mathbf{m}$ pourra être tronquée pour arriver à un mot de longueur exactement n.
**** Etape 2 ****
tant que *le dernier symbole de* $\mathbf{q}$ *est le plus grand symbole de* Σ **faire**
  Ôter ce symbole de $\mathbf{q}$.
**** Etape 3 ****
Remplacer le dernier symbole de $\mathbf{q}$ par le symbole qui suit dans Σ.
retourner $\mathbf{q}$.

Q23. Donner l'indice dans $\mathbf{m}$ de la i^e lettre de $\mathbf{q}$ en fonction de i et $|m|$.

Q24. Pour $\Sigma = \{0, 1\}$, on donne le mot de Lyndon $\mathbf{m} = 00111$ et $n = 9$. Donner le mot de Lyndon généré par l'algorithme, en déroulant les différentes étapes produites par l'algorithme permettant d'aboutir au mot de Lyndon.

L'**algorithme 1** peut être utilisé pour générer tous les mots de Lyndon de longueur au plus n. Pour ce faire, on part du plus petit symbole de Σ et on itère les trois étapes (1-2-3) de l'algorithme jusqu'à arriver au mot vide.

Q25. Partant de $\mathbf{m} = 0$ et toujours pour $\Sigma = \{0, 1\}$, construire par l'algorithme tous les mots de Lyndon de longueur au plus 4.

Q26. Donner la complexité de l'**algorithme 1** au pire des cas en nombre d'ajouts ou de suppressions de caractères.

III.1.3 - Factorisation de mots de Lyndon

Définition 10 (Factorisation de Lyndon)**.**
Soit $\mathbf{m} \in \Sigma^*$. *Une factorisation de* $\mathbf{m}$ *est une suite* $\mathbf{m}_1, \dots \mathbf{m}_l$ *de mots de Lyndon telle que* $m = \mathbf{m}_1 \cdots \mathbf{m}_l$ *avec* $\mathbf{m}_1 \geq \mathbf{m}_2 \cdots \geq \mathbf{m}_l$.

On admet le résultat suivant :

Théorème 1 (Factorisation d'un mot)**.**
Tout mot $\mathbf{m} \in \Sigma^*$ *admet une unique factorisation de Lyndon.*

L'**algorithme 2** propose une méthode de factorisation d'un mot $\mathbf{m}$ (on ne demande pas de le justifier). Le principe est d'itérer sur la chaîne des symboles de $\mathbf{m}$ pour trouver le plus grand mot de Lyndon possible. Lorsqu'un tel mot est trouvé, il est ajouté à la liste $\mathcal{L}$ des facteurs de $\mathbf{m}$ et la recherche est poursuivie sur la sous-chaîne restante.

Q27. En utilisant l'**algorithme 2**, écrire une fonction `factorisation m` qui réalise la factorisation d'un mot $\mathbf{m}$ donné. Cette fonction fera appel à une ou des fonction(s) récursive(s).

Algorithme 2 : Algorithme de factorisation d'un mot de Lyndon

Données : $\mathbf{m} \in \Sigma^*$ un mot de Lyndon.
Résultat : la liste $\mathcal{L}$ des mots de Lyndon décroissants de la factorisation de $\mathbf{m}$.

```
L ← []
j ← 1
k ← 0
tant que j ≤ |m| faire
    si j = |m| ou m_k > m_j alors
        p = m_0 ··· m_{j−k−1}
        Ajouter p à L
        Supprimer p de m
        k ← 0
        j ← 1
    sinon
        si m_k = m_j alors
            k ← k + 1
            j ← j + 1
        sinon
            k ← 0
            j ← j + 1
retourner L.
```

III.2 - Mots de de Bruijn

III.2.1 - Définition

Définition 11 (Mot de de Bruijn)**.**
Un mot de de Bruijn d'ordre n sur Σ *est un collier qui contient tous les mots de* Σ^n *une et une seule fois.*

Par exemple, pour $\Sigma = \{a, b, c\}$ et $n = 2$, $\mathbf{m} = abacbbcca$ est un mot de de Bruijn puisqu'il contient une unique fois chaque mot de longueur 2 sur Σ, le mot 'aa' étant obtenu par circularité de $\mathbf{m}$.

Q28. Donner la longueur d'un mot de de Bruijn en fonction de n et du nombre k de symboles de Σ.

III.2.2 - Graphe de de Bruijn

Définition 12 (Graphe de de Bruijn)**.**
Le graphe de de Bruijn d'ordre n sur Σ *est le graphe orienté* $\mathcal{B}(k, n) = (V, E)$ *où :*

- $V = \Sigma^n$ *est l'ensemble des sommets du graphe ;*
- $E = \{(a\mathbf{m}, \mathbf{m}b), a, b \in \Sigma, \mathbf{m} \in \Sigma^{n-1}\}$ *est l'ensemble des arcs orientés du graphe.*

On value les arcs E par le dernier symbole du noeud terminal de chaque arc : ainsi $(a\mathbf{m}, \mathbf{m}b) \in E$ *est étiqueté par b.*

Dans ce graphe, certains arcs ont pour sommet initial et terminal un même sommet de V. Ces arcs sont appelés des boucles (**figure 1**).

Q29. Donner l'ensemble des mots de longueur 3 sur $\Sigma = \{0, 1\}$. En déduire de graphe de de Bruijn $\mathcal{B}(2, 3)$ associé.

Q30. Montrer que le degré entrant et le degré sortant de chaque sommet est égal à k.

Q31. En déduire le nombre d'arcs orientés $|E|$ en fonction de k et n.

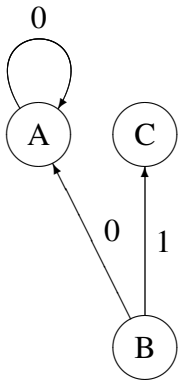


Figure 1 - Exemple de graphe avec boucle sur le sommet A.

Définition 13 (Successeur, prédécesseur)**.**
Soit $G = (V, E)$ un graphe orienté. $v \in V$ est un successeur (respectivement prédécesseur) de $u \in V$ si $(u, v) \in E$ (resp. $(v, u) \in E$).

Q32. Soient $\mathcal{B}(k, n) = (V, E)$ et $\mathbf{m} \in V$. Montrer que tous les prédécesseurs de $\mathbf{m}$ ont le même ensemble de successeurs.

Q33. Soit $\mathbf{p} = (p_0 \cdots p_{n-1}) \in V$. Donner l'ensemble des sommets $\mathbf{m}$ tels que $(\mathbf{p}, \mathbf{m}) \in E$.

On suppose disposer de $\mathcal{B}(k, n)$ et on souhaite construire $\mathcal{B}(k, n+1)$ à partir de ce graphe.

Q34. Proposer une méthode pour construire les sommets de $\mathcal{B}(k, n+1)$ à partir des arcs de $\mathcal{B}(k, n)$. Donner le sommet créé dans $\mathcal{B}(2, 4)$ à partir de l'arc (001,010) de $\mathcal{B}(2, 3)$.

On dit que deux arcs $e_1, e_2 \in E$ sont adjacents dans $\mathcal{B}(k, n)$ si ces deux arcs s'écrivent $e_1 = (\mathbf{m}, \mathbf{p})$ et $e_2 = (\mathbf{p}, \mathbf{q})$ pour $\mathbf{m}, \mathbf{p}, \mathbf{q} \in V$.

Q35. Proposer de même une construction des arcs de $\mathcal{B}(k, n+1)$ en fonction des arcs adjacents de $\mathcal{B}(k, n)$. Donner l'arc de $\mathcal{B}(2, 4)$ créé par les arcs adjacents de $\mathcal{B}(2, 3)$ (001,011) et (011,110).

III.2.3 - Construction des mots de de Bruijn

On propose ici trois algorithmes de construction de mots de de Bruijn.

III.2.3.1 - Construction à l'aide de $\mathcal{B}(\mathbf{k}, \mathbf{n})$

Les mots de de Bruijn d'ordre n sur Σ peuvent être construits en parcourant $\mathcal{B}(k, n)$.

Définition 14 (Circuit eulérien)**.**
Soit G un graphe orienté. Un circuit eulérien est un chemin dont l'origine et l'extrémité coïncident et passant une fois et une seule par chaque arête de G.

Q36. Construire $\mathcal{B}(2, 2)$ et trouver dans ce graphe un circuit eulérien. Vérifier que la concaténation des étiquettes lues au fil de ce circuit donne un représentant d'un mot de de Bruijn et donner son ordre sur $\{0, 1\}$.

On admet les résultats suivants :

(i). un graphe $G = (V, E)$ possède un circuit eulérien si et seulement si il est connexe et si, pour tout $v \in V$, le degré entrant de v et le degré sortant de v sont égaux.

(ii). un circuit eulérien dans le graphe $\mathcal{B}(k, n)$ correspond à un mot de de Bruijn.

Q37. En déduire qu'il existe au moins un mot de de Bruijn pour tout alphabet Σ et tout n.

Ainsi, la concaténation des étiquettes lues au fil d'un circuit eulérien de $\mathcal{B}(k, n)$ donne un représentant d'un mot de de Bruijn d'ordre $n + 1$ sur k symboles.

III.2.3.2 - Construction à l'aide de l'algorithme Prefer One

Pour construire les mots de de Bruijn d'ordre n sur $\Sigma = \{0, 1\}$, on peut également utiliser l'**algorithme 3**, dit algorithme `Prefer One`.

Algorithme 3 : Algorithme `Prefer One`

Données : n, Σ
Résultat : **m** mot de de Bruijn de longueur n sur Σ.
m ← suite de n zeros
STOP←`false`
tant que *STOP*=`false` **faire**
  `Etape 1`
  Ajouter un 1 à la fin de **m**.
  si *les n derniers symboles de* **m** *n'ont pas été rencontrés auparavant* **alors**
    Répeter `Etape 1`
  sinon
    Retirer le 1 ajouté à la fin de **m**.
    Passer à `Etape 2`
  `Etape 2`
  Ajouter un 0 à la fin de **m**.
  si *les n derniers symboles de* **m** *n'ont pas été rencontrés auparavant* **alors**
    Aller à l'`Etape 1`
  sinon
    STOP←`true`
retourner m.

Dans cet algorithme, "*les n derniers symboles de* **m** *n'ont pas été rencontrés auparavant*" signifie que le mot composé des n derniers symboles de **m** n'est pas un sous-mot de **m**, situé entre le premier et le $(|\mathbf{m}| - 1)^{\text{e}}$ symbole de **m**.

Q38. Appliquer l'algorithme au cas $n = 3$ et $\Sigma = \{0, 1\}$. Écrire les valeurs successives de **m** au cours de l'exécution.

III.2.3.3 - Construction à l'aide de la relation aux mots de Lyndon

La troisième construction utilise les notions de collier et de mots de Lyndon.

Q39. Donner, pour $k = 2$ et $n = 4$, les classes d'équivalence de tous les mots binaires de longueur 4 pour la relation C. En déduire les colliers correspondants.

Q40. En déduire les mots de Lyndon de longueur 4 pour $\Sigma = \{0, 1\}$.

Plus généralement, les mots de de Bruijn et de Lyndon sont étroitement liés. On peut montrer que si l'on concatène, dans l'ordre lexicographique, les mots de Lyndon sur k symboles dont la longueur divise un

entier n, alors on obtient le mot de de Bruijn le plus petit, pour l'ordre lexicographique, de tous les mots de de Bruijn de longueur n sur k symboles.

Q41. Déduire de la question précédente le plus petit mot de de Bruijn pour $n = 4$ et $\Sigma = \{0, 1\}$.

III.2.4 - Application

La soirée a été longue, vous rentrez chez vous mais, au pied de votre immeuble, vous vous trouvez confronté à un sérieux problème : vous avez complètement oublié le code d'entrée à n chiffres de la porte. On suppose que le digicode de l'appartement est composé de $1 \leq k \leq 10$ chiffres 0, 1,... $k-1$, qui constituent l'alphabet Σ.
Le digicode fonctionne de la façon suivante : vous tapez successivement sur les chiffres afin de composer un mot. À chaque nouveau symbole entré à partir du n-ième, le digicode teste le mot constitué par les n derniers chiffres pour voir s'il correspond au code secret.
Ainsi, par exemple pour $n = 4$, si vous tapez la séquence 021201, le digicode teste successivement 0212, 2120 et 1201.
Étant pressé de regagner votre lit, vous cherchez à taper un minimum de touches pour ouvrir la porte. On note $\tilde{n}$ la longueur de la plus petite séquence de frappe de touches qui vous permet de rentrer dans l'immeuble.

Q42. Donner un encadrement de $\tilde{n}$:
- pour la borne supérieure, on considèrera que l'on met bout à bout tous les mots possibles de n chiffres construits sur Σ ;
- pour la borne inférieure, on cherchera un mot sans redondance, c'est-à-dire qui contient une et une seule fois chaque mot de n chiffres.

Q43. Expliquer en une phrase en quoi les mots de de Bruijn peuvent vous aider. En vous inspirant de l'**algorithme 3**, donner une séquence la plus courte de chiffres à taper pour ouvrir à coup sûr la porte de votre immeuble, lorsque $n = 2$ et $k = 4$.

Q44. Comparer alors le nombre maximum de frappes de touches du digicode à effectuer en utilisant les mots de de Bruijn, avec celui calculé par la méthode brute. Calculer ces nombres pour :
- $k = 4$ et $n = 2$.
- $k = 10$ et $n = 4$.

Que conclure quant à l'efficacité de votre stratégie ?

FIN

IMPRIMERIE NATIONALE – 20 1154 – D'après documents fournis

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MP chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Logique, ensembles et applications
- Bases de Python : expressions, types, instructions, fonctions
- Langage OCaml : expressions, types, filtrage, récursivité
- Complexité des algorithmes itératifs et récursifs

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/e3a/mp