Concours Commun INP (CCINP), France

Session 2024, filière MP

Informatique

Durée : 4 heures

Ce que traite le sujet

- Partie I : coloration de graphes
- Partie II : satisfiabilité d'une formule propositionnelle
- Partie III : automates et reconnaissance de motifs

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/mp/2024/info

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

ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE MP

INFORMATIQUE

Durée : 4 heures

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, bleu clair ou turquoise, 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 indépendantes.

Partie I - Coloration de graphes

L'objectif de cette partie est de proposer une implémentation en **Python** d'une solution au problème de coloration d'un graphe.

I.1 - Définitions et propriétés

Soit $G = (S, A)$ un graphe fini non orienté avec S son ensemble de sommets et A son ensemble d'arêtes. On suppose que le graphe est simple c'est-à-dire qu'il ne comporte pas de boucles et que chaque paire de sommets est reliée par au plus une arête. On note n, le cardinal de l'ensemble S. Les sommets sont numérotés de 0 à $n-1$. Étant donné un entier naturel k, une k-coloration des sommets de G est une application $c : S \to \{0, 1, ..., k-1\}$ telle que pour chaque arête $\{x, y\}$ d'extrémités x et y, $c(x) \neq c(y)$. Si $c(x) = i$, on considèrera que la couleur i est affectée au sommet x. Si G admet une k-coloration, il est k-coloriable. On définit le nombre chromatique $\chi(G)$ d'un graphe G fini par $\chi(G) = \min\{k \in \mathbb{N}, \text{G est } k\text{-coloriable}\}$.
Une clique est un sous-ensemble de sommets du graphe, adjacents 2 à 2. On dit qu'un graphe est complet si il est une clique. On notera K_p le graphe complet à p sommets.
On pose $\omega(G) = \max\left\{p \in \mathbb{N} | K_p \text{ est une clique de G}\right\}$, avec $\mathbb{N}$ l'ensemble des entiers naturels.

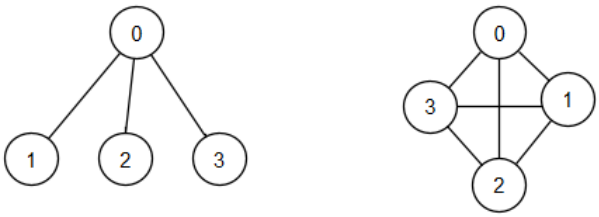


Figure 1 - de gauche à doite, le graphe G_1 et le graphe K_4

Q1. Le graphe G_1 de la **figure 1** ci-dessus est-il 2-coloriable ? Justifier votre réponse.

Q2. Pour un entier naturel $n \geq 1$, déterminer le nombre chromatique du graphe K_n.

Q3. Montrer que pour tout graphe G à n sommets, on a $\omega(G) \leq \chi(G) \leq n$.

I.2 - Algorithmique et programmation en Python (Informatique Commune)

La coloration d'un graphe G avec $\chi(G)$ couleurs est un problème complexe. Dans cette sous-partie, nous présentons une heuristique permettant de construire une coloration d'un graphe donné.
Dans la suite, on implémente un graphe par **un dictionnaire (type `dict` en Python), contenant les listes d'adjacence des sommets**. Les clés du dictionnaire sont les numéros des sommets et la valeur correspondant à la clé `i` du dictionnaire est la liste d'adjacence du sommet numéro `i`.
Notons qu'il serait ici possible d'implémenter le graphe par des listes d'adjacence dans un tableau, dans la mesure où les sommets sont numérotés de 0 à $n-1$.

Q4. Définir en Python le dictionnaire `d1` correspondant au graphe G_1.

Q5. Écrire une fonction Python `degres_sommets(d)` qui prend en paramètre un dictionnaire `d` représentant un graphe et renvoie une liste de couples `(di,i)`, `i` étant le numéro d'un sommet parcourant l'ensemble $\{0, ..., n-1\}$ et `di` le degré du sommet `i`.

Q6. On suppose qu'on dispose d'une fonction Python `tri(l)` qui trie une liste de couples `l` dans l'ordre décroissant par rapport à la première composante du couple. En déduire une fonction `tri_degres(d)` qui prend en paramètre un dictionnaire `d` représentant un graphe et renvoie une liste contenant les numéros des sommets, triés par degrés décroissants.

Q7. Écrire une fonction Python `test(d,c)` qui prend en paramètre un dictionnaire `d` représentant un graphe G, un dictionnaire `c` dont les clés représentent les sommets du graphe G et les valeurs, leurs couleurs. La fonction renvoie `True` si `c` est une 2-coloration pour G.

On considère ci-dessous, l'algorithme de coloriage de Welsh-Powel.

Algorithme 1 : Welsh-Powel (coloration de graphe)

Entrée : un graphe `G` à `n` sommets
Sortie : liste d'entiers contenant en position `i` la couleur du sommet numéro `i`

```
1 Début
2     Ordonner les sommets selon les degrés décroissants dans une liste li ;
3     colorie : dictionnaire vide qui à terme, associera à chaque clé i, la couleur du sommet i ;
4     Tant qu' il reste des sommets à colorier faire
5         Chercher dans li le premier sommet non colorié et le colorier avec la plus petite couleur
            c non utilisée ;
6         Colorier avec cette même couleur, en respectant leur ordre dans li, tous les sommets
            non coloriés et non adjacents à des sommets de couleur c ;
7     Fin
8     Retourner colorie
9 Fin
```

Q8. Que contient `colorie` si on déroule l'algorithme de coloriage ci-dessus avec le graphe G_1 en entrée ?

Q9. Écrire une fonction Python `adjacent(d,dc,s,c)` qui prend en paramètre un graphe représenté par un dictionnaire `d`, un dictionnaire `dc` contenant la couleur des sommets coloriés, le numéro d'un sommet `s`, une couleur `c` et renvoie `True` si le sommet `s` est adjacent à l'un des sommets de couleur `c`, `False` sinon.

Q10. Proposer une implémentation en Python de l'algorithme de Welsh-Powel.

Application
Le tableau ci-dessous représente les liens d'amitiés entre huit étudiants : Alice (A), Béatrice (B), Carl (C), David (D), Eloïs (E), Fanny (F), Gary(G) et Hedge (H).

Prénom	A	B	C	D	E	F	G	H
Ami·e avec	B,C,G	A,C,E,F	A, B	E,F	B,D,F	B,D,E,H	A,H	F,G

On souhaite créer des groupes de travail. Dans le contexte de l'application, un groupe contient au moins 2 étudiants tel que chaque étudiant soit dans un groupe différent de celui de ses amis.
Q11. Modéliser la situation par un graphe et en déduire une solution.

Partie II - Satisfiabilité d'une formule propositionnelle

Le langage utilisé dans cette partie est OCaml.
Une formule propositionnelle est construite à l'aide de constantes propositionnelles, de variables propositionnelles et de connecteurs logiques. Les connecteurs logiques seront notés $\neg$ (négation), $\wedge$ (conjonction), $\vee$ (disjonction). Dans cette partie, on étudie le problème de satisfiabilité d'une formule et son application à la détermination d'une conséquence logique entre 2 formules propositionnelles. Le problème CNF-SAT est défini de la façon suivante. Étant donné une formule sous forme normale conjonctive, admet-elle un modèle, c'est-à-dire une valuation des variables, qui rende la formule vraie ? On souhaite écrire un programme qui teste si une valuation donnée rend une telle formule vraie.
Dans cette partie, on considère que si une formule contient n variables propositionnelles, elles seront désignées par $x_0, x_1, ..., x_{n-1}$.
On définit le type OCaml suivant :

```
type clause=Var of int|Non of clause |Ou of clause*clause
```

L'argument du constructeur `Var` correspond au numéro de la variable concernée.
Une formule sous forme normale conjonctive ayant m clauses sera implémentée par une liste de m clauses. Les tableaux seront implémentés par le module `Array` dont les éléments suivants pourront être utilisés :

- `type 'a array`, notations `[| |]`
- création d'un tableau : `make : int -> 'a -> 'a array`
- accès à l'élément d'indice `i` du tableau `t` : `t.(i)`
- modification de l'élément placé à l'indice `i` du tableau `t` : `t.(i) <- v`
- taille du tableau : `length : 'a array -> int`

Q12. Donner le code OCaml correspondant à la clause $c = (x_0 \vee x_1) \vee \neg x_2$.

Q13. Donner le code OCaml permettant de définir la formule : $f = (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$.

Q14. Écrire une fonction de signature `evalue_clause : clause -> bool array -> bool` qui prend en paramètre une clause et une valuation représentée par un tableau contenant à l'indice i, la valeur de vérité de la variable x_i et renvoie la valeur de vérité de la clause.

Q15. Écrire une fonction de signature `evalue_FNC : clause list -> bool array -> bool` qui prend en paramètre une clause et une valuation représentée par un tableau contenant à l'indice i, la valeur de vérité de la variable x_i et évalue une formule donnée sous forme normale conjonctive.

Q16. Quel résultat obtient-on avec la formule F et le tableau de valuations `[|false;true;true|]` ? Justifier.

Q17. On souhaite énumérer toutes les valuations possibles pour un nombre de variables fixé. Étant donné une valuation, on considérera que si la valeur `true` correspond à 1 et la valeur `false` correspond à 0, la valuation suivante correspond à l'ajout de 1 au nombre binaire associé. Ainsi, la valuation suivante de `[|false;true;false|]` est `[|false;true;true|]`. On considère que la valuation suivante de `[|true;true;true|]` n'existe pas.
Écrire une fonction de signature `suivant : bool array -> bool` qui prend en paramètre un tableau de booléens, lui attribue la valuation "suivante" si possible et renvoie `true` ; sinon renvoie `false`.

Q18. En déduire une fonction de signature `satisfiable : clause list -> int -> bool` qui prend en paramètre une formule en forme normale conjonctive, son nombre de variables et renvoie `true` si il existe une valuation qui rend la formule vraie, `false` sinon.

Q19. Quelle est la complexité en temps de cette fonction par rapport aux paramètres d'entrée ?

Q20. Proposer une stratégie de retour sur trace pour résoudre le problème de satisfiabilité d'une formule.

Conséquence logique entre 2 formules

Définition

Une formule ϕ est une conséquence logique d'un ensemble fini de n formules $\Gamma = \{F_1, ..., F_n\}$, n étant un entier naturel supérieur ou égal à 1, si tout modèle de ϕ est un modèle de Γ. On note $\Gamma \vDash \phi$. On admettra que toute formule admet une formule équivalente sous forme normale conjonctive.

Q21. Déduire de la fonction précédente, un algorithme en pseudo-code permettant de déterminer si une formule F est une conséquence logique d'un ensemble de formules $\Gamma : F_1, ..., F_n$.

Q22. Afin de déterminer si $\Gamma \vDash \phi$, on peut prouver le séquent $\Gamma \vdash \phi$. Justifier cette méthode, puis construire un arbre de preuve qui démontre le séquent $\Gamma : P \rightarrow Q, Q \rightarrow R, P \vdash P$ où P, Q, R désignent des variables propositionnelles représentant des formules logiques, à partir des règles d'inférence de la déduction naturelle ; les règles et notations utilisées seront clairement mentionnées.

Partie III - Automates et reconnaissance de motifs

Dans cette partie, le langage utilisé est OCaml.

On s'intéresse à la reconnaissance de motifs dans un texte en utilisant une méthode naïve, puis des automates.

III.1 - Autour de l'algorithme naïf de recherche de caractères

Définitions

Un alphabet A est un ensemble fini non vide d'éléments appelés lettres. Un mot défini sur A est une suite finie d'éléments de A. Le mot vide sera noté ϵ. L'ensemble des mots sur l'alphabet A est noté A^*. La longueur d'un mot x, notée $|x|$ est le nombre de lettres de ce mot. Pour i allant de 0 à $|x|-1$, x_i désignera la lettre en position i . Ainsi, aab est un mot de longueur 3, sur tout alphabet contenant les lettres a et b.
Un mot x est un facteur d'un mot y si il existe 2 mots u et v tels que $y = uxv$. Dans ce cas, on dit qu'il y a une occurrence de x dans y. Quand $u = \epsilon$, x est un préfixe de y et quand $v = \epsilon$, x est un suffixe de y.
Soit x un mot non vide. Un entier $0 < p \leq |x|$ est une période de x si $x_i = x_{i+p}$ pour $0 \leq i \leq |x| - p - 1$. On définit la période de x comme étant la plus petite des périodes.
Un bord d'un mot non vide x est un facteur de x, différent de x et qui est à la fois un préfixe et un suffixe de la chaîne de caractères. Par exemple, ϵ, a,aa,$aabaa$ sont les bords du mot $aabaabaa$. On note $Bord(x)$ le plus long bord de x. On constate que lorsqu'il est défini, le bord d'un bord d'un mot x est aussi un bord de x. On le note $Bord^2(x)$. On définit ainsi récursivement $Bord^n(x)$ avec n un entier naturel.

En OCaml, les fonctions suivantes pourront être utilisées sur le type `string` :
- `String.length` : longueur de la chaîne de caractères
- `s.[i]` : accès à la lettre d'indice i de la chaîne s
- `^` : opérateur concaténation

Q23. Justifier le fait que tout mot non vide possède au moins une période.

Q24. Écrire une fonction de signature `occurrence : string -> string -> bool` qui prend en paramètre 2 chaînes de caractères `x` , `y` et renvoie `true` si il existe une occurrence de `x` dans `y`, `false` sinon.

Q25. Quelle est la complexité en temps de la fonction `occurrence` en fonction de la longueur des chaînes de caractères en entrée ?

Q26. Déterminer la période du mot $x = aabaabaa$ défini sur l'alphabet $A = \{a, b\}$.

Q27. Écrire une fonction de signature `periode : string -> int` qui renvoie la période d'une chaîne de caractères.

Q28. Soit x une chaîne de caractères non vide et p un entier naturel tel que $0 < p \leq |x|$. Montrer que p est une période de x si et seulement si il existe 3 chaînes de caractères u, v, w telles que $x = uw = wv$ et $|u| = |v| = p$.

Q29. Soit x une chaîne de caractères non vide et n le plus grand entier tel que $Bord^n(x)$ est bien défini. On a donc $Bord^n(x) = \epsilon$. Montrer par récurrence sur la longueur des mots que $Bord(x), Bord^2(x), ..., Bord^n(x)$ est la suite des bords de x dans l'ordre décroissant de leur longueur et que $|x| - |Bord(x)|, |x| - Bord^2(x)|, ..., |x| - |Bord^n(x)|$ est la suite des périodes de x dans l'ordre croissant.

Q30. Expliquer brièvement comment la connaissance des bords peut permettre d'améliorer la complexité en temps dans le pire cas, de l'algorithme naïf de recherche d'une occurrence d'un mot.

III.2 - Localisation des occurrences d'un motif à l'aide d'un automate

Un automate déterministe M sur l'alphabet A est composé d'un ensemble d'états finis Q, d'un état initial q_0, d'un ensemble d'états terminaux $T \subseteq Q$ et d'une fonction de transitions $\delta : Q \times A \to Q$. On le désigne par un quintuplet (Q, A, q_0, T, δ). On dit qu'il est complet si δ est définie pour chaque élément de $Q \times A$.
Certains automates peuvent être utilisés comme machine de recherche pour le traitement séquentiel de textes. Étant donné un alphabet A, un motif X représente un langage non vide ne contenant pas le mot vide. Le problème de la recherche de motifs est celui de la localisation d'occurrences de mots du langage dans d'autres mots. On suppose qu'on dispose d'un automate déterministe M complet qui reconnaît le langage A^*X, autrement dit, l'automate M reconnaît les mots qui ont comme suffixe un mot de X.
L'algorithme suivant permet de localiser les mots de X reconnus dans un texte t, considéré comme une chaîne de caractères.
Dans la suite, l'automate M de la **figure 2** ci-après sera cité en exemple. Il est défini sur l'alphabet $A = \{a, b, c\}$

Algorithme 2 : reconnaissance de mots dans un texte

```
1  Cherche(M,t) ;
2  Début
3      e← initial[M] ;
4      lst← liste vide ;
5      Pour chaque lettre λ de t prise dans l'ordre croissant de leurs indices faire
6          e ← δ(e,λ) ;
7          Si e est un état terminal alors
8              ajouter à lst l'indice de λ dans t ;
9          Fin
10     Fin
11     Retourner lst ;
12 Fin
```

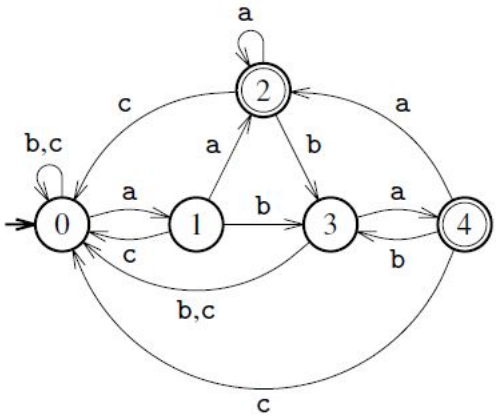


Figure 2 - Automate M

On définit en OCaml un type automate par :

```
type auto={etats: int list; alphabet : char list; initial: int;
transition: int -> char ->int; final : int list};;
```

Q31. Créer une variable `automate` qui représente l'automate M de la **figure 2**.

Q32. L' automate M est-il émondé ? Justifier.

Q33. Présenter les premières étapes de l'algorithme d'élimination des états appliqué à l'automate M. On éliminera l'état 0.

Q34. Que représentent les indices mémorisés dans la liste `lst` de l'algorithme `cherche` ? Que contient la liste `l` de l'algorithme `cherche` avec l'automate ci-dessus et $t = cabaac$?

Q35. Écrire une fonction de signature `cherche : auto -> string -> int list` qui implémente l'algorithme `cherche`.

Q36. Montrer la correction de l'algorithme `cherche`.

FIN

IMPRIMERIE NATIONALE – 24 1094 – 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

- Langage OCaml : expressions, types, filtrage, récursivité
- Bases de Python : expressions, types, instructions, fonctions
- 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/ccinp/mp