ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DES TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE
(Filière T.S.I.)

CONCOURS D'ADMISSION 2003

ÉPREUVE D'INFORMATIQUE

Filière MP
(Durée de l'épreuve : 3 heures)

Sujet mis à la disposition des concours Cycle International, ENSTIM et TPE-EIVP.

Les candidats et les candidates sont priés de mentionner de façon apparente sur la première page de la copie : « INFORMATIQUE – Filière MP »

RECOMMANDATIONS AUX CANDIDATS ET CANDIDATES

- **L'énoncé de cette épreuve, y compris cette page de garde, comporte 8 pages.**
- **Si, au cours de l'épreuve, un candidat ou une candidate repère ce qui lui semble être une erreur d'énoncé, il ou elle le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il ou elle a décidé de prendre.**
- **Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.**
- **Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.**
- **L'utilisation d'une calculatrice ou d'un ordinateur est interdite.**

COMPOSITION DE L'ÉPREUVE

L'épreuve comporte deux parties indépendantes :

- un exercice de logique des propositions, page 2, à résoudre en 45 mn environ ;
- un problème d'algorithmique, pages 3 à 8, à résoudre en 135 mn environ.

Téléchargé gratuitement sur www.Doc-Solus.fr .

1. Exercice de logique – 45 mn environ

Logique propositionnelle tri-valuée

On désire étendre la logique propositionnelle bivaluée classique (notée par la suite $\mathcal{L}_2$) de sorte à prendre en compte, en plus de V (vrai) et F (faux), une troisième valeur de vérité, notée I (pour indéterminé).

La **logique de Lukasiewicz**, notée $\mathcal{L}_3$, est une telle logique tri-valuée dans laquelle le connecteur de négation ($\neg$), le connecteur de conjonction ($\wedge$), le connecteur de disjonction ($\vee$) et le connecteur d'implication ($\Rightarrow$) sont définis par les tables de vérités suivantes :

x	$\neg x$
V	F
F	V
I	I

$\neg x$

x \ y	V	F	I
V	V	F	I
F	F	F	F
I	I	F	I

$x \wedge y$

x \ y	V	F	I
V	V	V	V
F	V	F	I
I	V	I	I

$x \vee y$

x \ y	V	F	I
V	V	F	I
F	V	V	V
I	V	I	V

$x \Rightarrow y$

On dit que deux propositions sont **équivalentes** dans une logique donnée $\mathcal{L}$ si elles ont même table de vérité dans cette logique.

❒ 1 – Indiquer une proposition P_1 équivalente dans $\mathcal{L}_2$ à la proposition « $x \vee y$ » et n'utilisant pas d'autres connecteurs que $\neg$ et $\wedge$. Les propositions P_1 et « $x \vee y$ » sont-elles équivalentes dans $\mathcal{L}_3$?

❒ 2 – Indiquer une proposition P_2 équivalente dans $\mathcal{L}_2$ à la proposition « $x \Rightarrow y$ » et n'utilisant pas d'autres connecteurs que $\neg$ et $\vee$. Les propositions P_2 et « $x \Rightarrow y$ » sont-elles équivalentes dans $\mathcal{L}_3$?

❒ 3 – Établir (en détaillant son obtention) la table de vérité dans $\mathcal{L}_3$ de la proposition P_3 suivante :

$$((x \Rightarrow y) \wedge ((\neg x) \Rightarrow y)) \Rightarrow y.$$

Cette proposition est-elle une tautologie dans $\mathcal{L}_3$ (la définition d'une tautologie dans la logique $\mathcal{L}_3$ reste la même que dans la logique $\mathcal{L}_2$) ?

❒ 4 – On appelle **littéral** une variable propositionnelle ou sa négation. Dans $\mathcal{L}_2$, toute proposition logique admet une forme normale disjonctive (c'est-à-dire une disjonction de conjonctions de littéraux) qui lui est équivalente. Établir une proposition P_4 sous forme normale disjonctive équivalente dans $\mathcal{L}_2$ à la proposition « $(x \vee y) \Rightarrow z$ ». Les propositions P_4 et « $(x \vee y) \Rightarrow z$ » sont-elles équivalentes dans $\mathcal{L}_3$?

❒ 5 – Peut-on faire un raisonnement par contraposition dans $\mathcal{L}_3$ (on justifiera la réponse) ?

❒ 6 – On définit un ordre sur les valeurs F, I et V par : $F < I < V$. On considère une logique $\mathcal{L}$ tri-valuée pour laquelle simultanément :

- $\mathcal{L}$ est une extension de $\mathcal{L}_2$ pour les quatre connecteurs usuels ($\neg$, $\wedge$, $\vee$, $\Rightarrow$) ; autrement dit, les restrictions aux valeurs V et F des tables de vérité de $\mathcal{L}$ donnent les tables de vérité de $\mathcal{L}_2$;
- le connecteur $\wedge$ est commutatif ;
- la proposition « $x \vee y$ » est équivalente à P_1 ;
- la proposition « $x \Rightarrow y$ » est équivalente à P_2 ;
- pour toute valeur de y, la fonction $x \mapsto x \wedge y$ croît au sens large avec x ;
- les propositions « $\neg (\neg x)$ » et « x » sont équivalentes.

Combien y a-t-il de telles logiques (on justifiera la réponse) ?

Téléchargé gratuitement sur www.Doc-Solus.fr .

2. Problème d'algorithmique – 135 mn environ

Coloration d'un graphe

Préliminaire concernant la programmation : il faudra écrire des fonctions ou des procédures à l'aide d'un langage de programmation qui pourra être soit **Caml**, soit **Pascal**, tout autre langage étant exclu. **Indiquer en début de problème le langage de programmation choisi ; il est interdit de modifier ce choix au cours de l'épreuve.** Certaines questions du problème sont formulées différemment selon le langage de programmation ; cela est indiqué chaque fois que c'est nécessaire. Par ailleurs, lorsqu'un candidat ou une candidate écrira une fonction ou une procédure en langage de programmation, il ou elle précisera si nécessaire le rôle des variables locales et pourra définir des fonctions ou procédures auxiliaires qu'il ou elle explicitera. Enfin, lorsque le candidat ou la candidate écrira une fonction ou une procédure, il ou elle pourra faire appel à une autre fonction ou procédure définie dans les questions précédentes.

Définitions et notations

On utilisera dans tout le problème les notations et définitions suivantes :

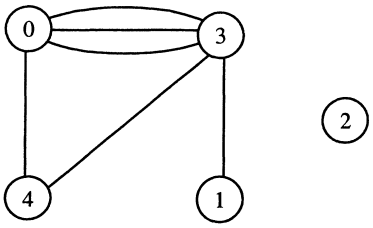
- une paire est un ensemble composé de deux éléments a et b **distincts** et est notée $\{a, b\}$ (ce qui est égal à $\{b, a\}$) ;
- un **graphe** G est composé d'un ensemble $S(G)$ d'éléments appelés **sommets** et d'une liste $A(G)$ de paires de sommets ; les éléments de $A(G)$ sont appelés **arêtes** et ne sont pas nécessairement tous distincts ;
- le cardinal de $S(G)$ est noté $n(G)$ et on a toujours : $S(G) = \{0, 1, \ldots, n(G) - 1\}$; un graphe est donc entièrement décrit par le couple $(n(G), A(G))$;
- deux sommets s et t de G sont dits **voisins** si $\{s, t\}$ est une arête de G ;
- un même identificateur écrit dans deux polices de caractères différentes désignera la même entité mais du point de vue mathématique pour une police (en italique) et du point de vue informatique pour l'autre (en romain) ; par exemple, `G` sera la représentation informatique du graphe G.

Une représentation graphique d'un graphe G consiste à associer à chaque sommet de G un cercle contenant le numéro du sommet et à tracer une ligne entre deux cercles représentant des sommets si les sommets correspondants forment une arête de G.

Exemple introductif : un graphe (nommé *Gex1*) et une représentation de *Gex1*

On considère le graphe *Gex1* dessiné à droite et dont les caractéristiques sont :
$n(Gex1) = 5$
$A(Gex1) = \{\{0, 4\}, \{4, 3\}, \{0, 3\}, \{1, 3\}, \{0, 3\}, \{3, 0\}\}$.

Il y a trois arêtes entre le sommet 0 et le sommet 3.
Le sommet 0 a pour voisins les sommets 3 et 4.
Le sommet 1 a pour voisin le sommet 3.
Le sommet 2 n'a pas de voisin.
Le sommet 3 a pour voisins les sommets 0, 1 et 4.
Le sommet 4 a pour voisins les sommets 0 et 3.



Le graphe *Gex1*

Indications pour la programmation

Caml : On définit les types `Arete` et `Graphe` par :

```
type Arete = {a : int; b : int};;
type Graphe = {n : int; A: Arete list};;
```

Ainsi, le graphe *Gex1* de l'exemple introductif est défini par :

```
let Gex1 = {n = 5; A = [{a = 0; b = 4}; {a = 4; b = 3}; {a = 0; b = 3};
                        {a = 1; b = 3}; {a = 0; b = 3}; {a = 3; b = 0}]};;
```

Les variables de type `Arete` ou de type `Graphe` sont des enregistrements ; un enregistrement contient des champs ; par exemple, une variable de type `Arete` contient les champs `a` et `b`. On peut accéder à un champ d'une variable de type enregistrement en faisant suivre le nom de cette variable d'un point puis du nom du champ considéré ; par exemple, on accède au champ `n` du graphe `Gex1` par `Gex1.n` (qui vaut 5).

Téléchargé gratuitement sur www.Doc-Solus.fr .

Pascal : Dans tout le problème, on supposera qu'on écrit les différentes fonctions dans un fichier contenant les définitions suivantes :

```
const
   MAX_SOMMETS = 100;
   MAX_ARETES  = 1000;

type Arete = RECORD
            a : integer;
            b : integer;
     end;

type Graphe = RECORD
            n           : integer;
            nb_aretes   : integer;
            A           : array[0..MAX_ARETES - 1] of Arete;
     end;

type Table = RECORD
            nb_donnees  : integer;
            cases       : array[0..MAX_SOMMETS - 1] of integer;
     end;
```

Les types `Arete`, `Graphe` et `Table` sont des types pour des enregistrements (`RECORD`). Un enregistrement contient des champs (quelquefois aussi appelés des membres) ; par exemple, une variable de type `Arete` contient les champs `a` et `b` ; on peut accéder à un champ d'une variable de type enregistrement en faisant suivre le nom de cette variable d'un point puis du nom du champ considéré, comme on le voit dans la définition de `Gex1` ci-dessous. Les variables de type enregistrement se manipulent comme toute autre variable : on peut définir des variables de type enregistrement, on peut affecter à une variable de type enregistrement la valeur d'une autre variable du même type, les variables de type enregistrement peuvent servir de paramètres à des fonctions ou procédures et peuvent être renvoyées par des fonctions ; en revanche, il est interdit de les comparer directement.

Ainsi, le graphe *Gex1* de l'exemple introductif est défini par :

```
Gex1.n := 5;
Gex1.nb_aretes := 6;
Gex1.A[0].a := 0;       Gex1.A[0].b := 4;
Gex1.A[1].a := 4;       Gex1.A[1].b := 3;
Gex1.A[2].a := 0;       Gex1.A[2].b := 3;
Gex1.A[3].a := 1;       Gex1.A[3].b := 3;
Gex1.A[4].a := 0;       Gex1.A[4].b := 3;
Gex1.A[5].a := 3;       Gex1.A[5].b := 0;
```

On supposera que les graphes traités n'ont jamais plus de `MAX_SOMMETS` sommets ou plus de `MAX_ARETES` arêtes.

Remarques importantes (langage Pascal) : lorsqu'on aura affaire à une variable `T` de type `Table`, le tableau `T.cases` servira à contenir un ensemble de données entières et on convient que le champ `T.nb_donnees` **devra toujours indiquer** le nombre de données de cet ensemble (et non la longueur totale `MAX_SOMMETS` du tableau `T.cases`) ; les données contenues dans `T.cases` se trouveront donc toujours dans les cases indicées de `0` à `T.nb_donnees - 1`. Par ailleurs, pour une variable `G` de type `Graphe` correspondant à un graphe G, `G.n` contiendra le nombre $n(G)$ de sommets de G, `G.nb_aretes` contiendra le nombre d'arêtes de G et le tableau `G.A` contiendra les arêtes de G entre les indices `0` et `G.nb_aretes - 1`.

Téléchargé gratuitement sur www.Doc-Solus.fr .

I – Détermination des voisins des sommets

❒ 7 – Il s'agit dans cette question de programmer une fonction qui insère une donnée entière dans une suite triée d'entiers distincts à condition que cette nouvelle donnée ne figure pas déjà dans la suite.

Caml : Écrire une fonction `insere` telle que, si `L` est une liste d'entiers distincts triée par ordre croissant et `s` un entier quelconque, `insere L s` renvoie
- la liste d'entiers obtenue en insérant `s` dans `L` selon l'ordre croissant si la valeur `s` ne figurait pas dans `L`,
- la liste `L` si `s` figurait déjà dans `L`.

Pascal : Écrire une fonction `insere` telle que, lorsque
- `T` est une variable de type `Table` telle que `T.cases` contient `T.nb_donnees` entiers triés par ordre croissant et
- `s` est un entier,

`insere(T, s)` renvoie
- un résultat de type `Table` obtenu en insérant `s` dans `T.cases` selon l'ordre croissant si la valeur `s` ne figurait pas dans `T.cases` (on n'oubliera pas d'actualiser `T.nb_donnees`),
- `T` si `s` figurait déjà dans `T.cases`.

❒ 8 – On s'intéresse à la complexité de la fonction `insere`.

Caml : Indiquer, en fonction de la longueur de `L`, la complexité dans le pire des cas de l'algorithme utilisé pour la fonction `insere` de la question ❒ 7.

Pascal : Indiquer, en fonction de `T.nb_donnees`, la complexité dans le pire des cas de l'algorithme utilisé pour la fonction `insere` de la question ❒ 7.

❒ 9 – Il s'agit dans cette question d'écrire une fonction qui donne la liste triée des voisins d'un sommet donné dans un graphe donné.

Caml : Écrire en Caml une fonction `voisins` telle que `voisins G s` renvoie la liste triée par ordre croissant des voisins du sommet s dans le graphe G. On utilisera pour cela la fonction `insere`.

Pascal : Écrire en Pascal une fonction `voisins` telle que `voisins(G, s)` renvoie un résultat de type `Table` dont :
- le champ `cases` contient les voisins du sommet s dans le graphe G triés par ordre croissant,
- le champ `nb_donnees` contient le nombre de ces voisins.

On utilisera pour cela la fonction `insere`.

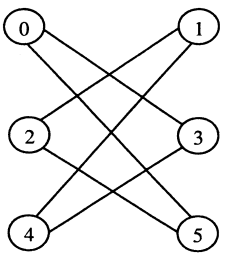
II – Un algorithme de bonne coloration d'un graphe

On appelle **coloration** d'un graphe G toute application c de l'ensemble S des sommets de G dans l'ensemble des entiers naturels non nuls. Pour tout sommet s, l'entier $c(s)$ s'appelle **couleur** de s pour la coloration c.

Une coloration c est une **bonne coloration** de G si pour toute arête $\{s, t\}$ de G, on a $c(s) \neq c(t)$; autrement dit, une coloration est une bonne coloration si les extrémités de toute arête sont de couleurs différentes.

On considère l'algorithme de bonne coloration suivant. Les couleurs disponibles sont les entiers naturels non nuls ; la plus petite couleur est ainsi la couleur 1. On colorie les sommets les uns après les autres, dans l'ordre 0, 1, 2 …, $n(G) - 1$; lorsqu'on considère un sommet s, on lui attribue la plus petite couleur disponible, c'est-à-dire la plus petite couleur qui n'est pas déjà la couleur d'un voisin de s.

❒ 10 – Que donne cet algorithme pour le graphe *Gex2* ci-contre ? Y a-t-il une autre bonne coloration de G utilisant moins de couleurs ?



Le graphe *Gex2*

Téléchargé gratuitement sur www.Doc-Solus.fr .

❒ 11 – Il s'agit d'écrire en langage de programmation l'algorithme de bonne coloration décrit plus haut. Pour déterminer cette bonne coloration, on utilisera $n(G)$ cases à valeurs entières d'un tableau dont les indices 0, 1, ..., $n(G) - 1$ correspondront aux sommets de G ; la case d'indice s contiendra après le déroulement de l'algorithme la couleur du sommet s.

Caml : Écrire en Caml une fonction `coloration` telle que `coloration G` renvoie le tableau des couleurs attribuées aux sommets du graphe G par l'algorithme décrit ci-dessus.

Pascal : On définit pour cette question un nouveau type par :

```
type Couleurs = array[0..MAX_SOMMETS - 1] of integer;
```

Écrire en Pascal une fonction `coloration` telle que `coloration(G)` renvoie le tableau des couleurs attribuées aux sommets du graphe G par l'algorithme décrit ci-dessus ; le résultat retourné par la fonction `coloration` sera de type `Couleurs`.

III – Définition du nombre chromatique de G

Soit G un graphe et p un entier strictement positif. On appelle **bonne p-coloration** de G une bonne coloration n'utilisant que des couleurs appartenant à l'ensemble $\{1, ..., p\}$. On introduit les notations suivantes :

- $BC(G, p)$ est l'ensemble des bonnes p-colorations de G ;
- $fc(G, p)$ est le cardinal de $BC(G, p)$;
- $EC(G) = \{p \in \mathbb{N}^* \text{ tel que } fc(G, p) \neq 0\}$.

❒ 12 – Montrer l'existence d'un unique entier $nbc(G)$ tel que :

$$EC(G) = \{p \in \mathbb{N}^* \text{ tel que } p \geq nbc(G)\}.$$

On dit que $nbc(G)$ est le **nombre chromatique** du graphe G : c'est le nombre minimum de couleurs permettant de colorier les sommets de G de sorte que deux sommets voisins n'aient pas la même couleur.

❒ 13 – Soit G un graphe n'ayant aucune arête. Déterminer $nbc(G)$ en fonction de $n(G)$ et, pour tout $p \in \mathbb{N}^*$, déterminer $fc(G, p)$ en fonction de $n(G)$ et p.

❒ 14 – Soit G un graphe tel que toute paire de sommets soit une arête. Déterminer $nbc(G)$ en fonction de $n(G)$ et, pour tout $p \in \mathbb{N}^*$, déterminer $fc(G, p)$ en fonction de $n(G)$ et p.

❒ 15 – On considère le graphe *Gex1* de l'exemple introductif. Déterminer $nbc(Gex1)$ et, pour tout $p \in \mathbb{N}^*$, déterminer $fc(Gex1, p)$ en fonction de p.

IV – Les applications H et K

On dit qu'un sommet d'un graphe est **isolé** s'il n'a aucun voisin. Par exemple, le sommet 2 est isolé dans le graphe *Gex1*.

❒ 16 – Étant donnés un graphe G et un sommet de G non isolé s, on note $prem\_voisin(G, s)$ le plus petit voisin de s dans G, c'est-à-dire le voisin de s qui a le plus petit numéro. Par exemple, $prem\_voisin(Gex1, 0)$ est le sommet 3.

Caml : Écrire en Caml une fonction `prem_voisin` telle que `prem_voisin G s` renvoie $prem\_voisin(G, s)$. Cette fonction ne prévoira pas le cas où s est un sommet isolé de G.

Pascal : Écrire en Pascal une fonction `prem_voisin` telle que `prem_voisin(G, s)` renvoie $prem\_voisin(G, s)$. Cette fonction ne prévoira pas le cas où s est un sommet isolé de G.

❒ 17 – On considère un graphe G possédant au moins une arête. On note $prem\_ni(G)$ le plus petit sommet non isolé du graphe G, c'est-à-dire le sommet non isolé qui a le plus petit numéro. Par exemple, $prem\_ni(Gex1)$ est le sommet 0.

Caml : Écrire une fonction `prem_ni` telle que `prem_ni G` renvoie $prem\_ni(G)$. Cette fonction ne prévoira pas le cas où G ne possède aucune arête.

Pascal : Écrire une fonction `prem_ni` telle que `prem_ni(G)` renvoie $prem\_ni(G)$. Cette fonction ne prévoira pas le cas où G ne possède aucune arête.

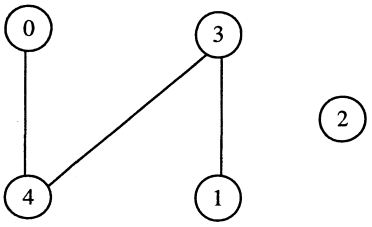
❒ 18 – Rappelons d'abord que dans la liste $A(G)$ des arêtes d'un graphe G, il est possible qu'une même arête soit répétée.

Étant donné un graphe G possédant au moins une arête, on pose :

Téléchargé gratuitement sur www.Doc-Solus.fr .

$s_1 = prem\_ni(G)$
$s_2 = prem\_voisin(G, prem\_ni(G))$.
Par exemple, pour le graphe *Gex1*, $s_1 = 0$ et $s_2 = 3$.
On note $H(G)$ le graphe obtenu à partir de G en supprimant toutes les arêtes entre s_1 et s_2.

Par exemple, le graphe $H(Gex1)$, représenté ci-contre, est caractérisé par :
$n(H(Gex1)) = 5$,
$A(H(Gex1)) = \{\{0, 4\}, \{4, 3\}, \{1, 3\}\}$.



Le graphe $H(Gex1)$

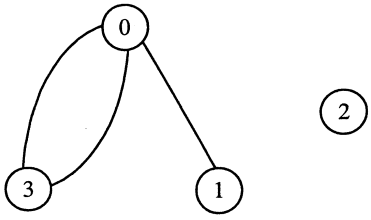
Caml : Écrire en Caml une fonction `H` telle que `H G` renvoie $H(G)$ (qui sera de type `Graphe`). Cette fonction ne prévoira pas le cas où G ne possède aucune arête.

Pascal : Écrire en Pascal une fonction `H` telle que `H(G)` renvoie $H(G)$ (qui sera de type `Graphe`). Cette fonction ne prévoira pas le cas où G ne possède aucune arête.

❒ 19 – On considère un graphe G possédant au moins une arête. On définit s_1 et s_2 comme dans la question précédente ; on peut remarquer la relation : $s_2 > s_1$. On construit alors un graphe noté $K(G)$ de la façon décrite ci-dessous.

- On construit $H(G)$;
- dans $H(G)$, on superpose s_1 et s_2 ; plus précisément,
  - on considère successivement chaque arête de la liste des arêtes de $H(G)$; pour chacune d'entre elles, on renumérote ses extrémités :
    - une extrémité de valeur strictement inférieure à s_2 est inchangée ;
    - une extrémité de valeur s_2 prend la valeur s_1 ;
    - une extrémité de valeur strictement supérieure à s_2 est décrémentée de 1 ;
  - on diminue de 1 le nombre de sommets du graphe.

Par exemple, $K(Gex1)$, représenté ci-contre, est décrit par :
$n(K(Gex1)) = 4$,
$A(K(Gex1)) = \{\{0, 3\}, \{3, 0\}, \{1, 0\}\}$.



Le graphe $K(Gex1)$

Caml : Écrire en Caml une fonction `K` telle que `K G` renvoie $K(G)$ (le résultat sera de type `Graphe`). Cette fonction ne prévoira pas le cas où G ne possède aucune arête.

Pascal : Écrire en Pascal une fonction `K` telle que `K(G)` renvoie $K(G)$ (le résultat sera de type `Graphe`). Cette fonction ne prévoira pas le cas où G ne possède aucune arête.

V – Fonction $fc(G, p)$ et polynôme chromatique

Soit G un graphe possédant au moins une arête et soit p un entier non nul.

❒ 20 – Montrer l'inclusion : $BC(G, p) \subset BC(H(G), p)$.

❒ 21 – Comparer les cardinaux de $BC(K(G), p)$ et de $BC(H(G), p) \setminus BC(G, p)$.

Téléchargé gratuitement sur www.Doc-Solus.fr .

❒ 22 – Montrer l'égalité : $fc(G, p) = fc(H(G), p) - fc(K(G), p)$.

❒ 23 – En utilisant la formule obtenue dans la question ❒ 22, décrire en termes simples un algorithme récursif de calcul de $fc(G, p)$; ici, on ne demande pas d'utiliser un langage de programmation.

❒ 24 – Prouver que l'algorithme de la question ❒ 23 se termine.

❒ 25 – Le graphe G étant fixé, montrer que la fonction qui associe à p la quantité $fc(G, p)$ est la restriction à $\mathbb{N}^*$ d'un polynôme en p de degré $n(G)$. On appelle **polynôme chromatique** de G et on note $Pc(G)$ ce polynôme.

VI – **Calcul du polynôme $Pc(G, p)$ et de $nbc(G)$**

On ne considérera dans cette partie que des polynômes à une variable et à coefficients entiers.

Caml : Un polynôme de degré d est représenté par un tableau à $(d + 1)$ cases, la case d'indice i contenant le coefficient du monôme de degré i. On pourra, pour connaître le degré d'un polynôme, utiliser la fonction de Caml `vect_length` qui, lorsqu'on invoque `vect_length T`, où `T` est un tableau, retourne le nombre de cases de ce tableau.

Pascal : On définit pour cette dernière partie un nouveau type par :

```
type Polynome = RECORD
                degre : integer;
                coeff : array[0..MAX_SOMMETS] of integer;
end;
```

On utilisera ce type pour représenter un polynôme dont le degré ne dépasse pas `MAX_SOMMETS`. Le champ `degre` contiendra le degré du polynôme et, pour `i` ≤ `degre`, `coeff[i]` contiendra le coefficient du monôme de degré i.

❒ 26 – Dans cette question, on va s'intéresser à la différence de deux polynômes. On ne traitera pas le cas où simultanément les deux polynômes auraient même degré et même coefficient du monôme de plus haut degré.

Caml : Écrire en Caml une fonction `difference` telle que, lorsque `P` et `Q` sont deux variables représentant des polynômes P et Q, `difference P Q` renvoie le polynôme $P - Q$ représenté par un tableau d'entiers comme indiqué dans l'introduction de cette partie.

Pascal : Écrire en Pascal une fonction `difference` telle que, lorsque `P` et `Q` sont deux variables de type `Polynome` représentant des polynômes P et Q, `difference(P, Q)` renvoie le polynôme $P - Q$, qui sera de type `Polynome`.

❒ 27 – Cette question consiste à calculer le polynôme chromatique $Pc(G)$.

Caml : Écrire en Caml une fonction `Pc` telle que `Pc G` renvoie le polynôme $Pc(G)$ représenté par le tableau de ses coefficients comme indiqué au début de cette partie.

Pascal : Écrire en Pascal une fonction `Pc` telle que `Pc(G)` renvoie le polynôme $Pc(G)$ sous la forme d'une variable de type `Polynome`.

❒ 28 – Dans cette question, il s'agit de calculer la valeur $P(x)$ d'un polynôme P en une valeur entière x. On n'utilisera pas de fonction prédéfinie du langage qui calculerait les puissances d'un entier. Les calculs seront faits avec les quatre opérations arithmétiques ordinaires. De plus, **la complexité de l'algorithme utilisé sera nécessairement du même ordre de grandeur que le degré du polynôme considéré**.

Caml : Écrire en Caml une fonction `eval` telle que `eval P x` renvoie $P(x)$.

Pascal : Écrire en Pascal une fonction `eval` telle que `eval(P, x)`, où `P` est de type `Polynome` et `x` de type `integer`, renvoie $P(x)$.

❒ 29 – Écrire dans le langage de programmation choisi une fonction qui calcule le nombre chromatique $nbc(G)$ d'un graphe G.

Téléchargé gratuitement sur www.Doc-Solus.fr .