A 2012 INFO. MP

ECOLE DES PONTS PARISTECH,
SUPAERO (ISAE), ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH,
MINES DE SAINT-ETIENNE, MINES DE NANCY,
TELECOM BRETAGNE, ENSAE PARISTECH (FILIERE MP)
ECOLE POLYTECHNIQUE (FILIERE TSI)

CONCOURS 2012

EPREUVE d'INFORMATIQUE

Filière : MP

Durée de l'épreuve : 3 heures.

L'utilisation d'une calculatrice est autorisée.

Sujet mis à la disposition des concours :
CYCLE INTERNATIONAL, ECOLES DES MINES, TELECOM SUDPARIS, TPE-EIVP.

L'énoncé de cette épreuve comporte 14 pages.

Les candidats sont priés de mentionner de façon apparente
sur la première page de la copie :
INFORMATIQUE - MP

Recommandations aux candidats

- **Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à 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.**

Composition de l'épreuve

L'épreuve comporte :

Exercice sur les langages rationnels

Un *alphabet* Σ est un ensemble fini non vide d'éléments appelés *lettres*. Un *mot* sur Σ est une suite finie de lettres de Σ ; la *longueur* d'un mot u est le nombre de lettres composant u ; le mot de longueur nulle est noté ε. On désigne par Σ^* l'ensemble des mots sur Σ, y compris le mot ε. Un *langage sur* Σ est une partie de Σ^*.

On dit qu'un mot f sur Σ, de longueur non nulle, est un *facteur* d'un mot u sur Σ s'il existe deux mots x et y sur Σ avec $u = xfy$; si x est le mot de longueur nulle, on dit que f est un *préfixe* de u ; si y est le mot de longueur nulle, on dit que f est un *suffixe* de u.

Un mot peut posséder plusieurs occurrences d'un même facteur f. Supposons que l'on ait $\Sigma = \{a, b\}$ et $f = aabaa$. Le mot *baabaabbaabaaa* contient *deux occurrences disjointes* du facteur f, de même que le mot *aabaaaabaaa*. Le mot *abaabaaabaabb* contient *deux occurrences non disjointes* du facteur f, de même que le mot *aabaabaa* ; dans ce dernier cas, la première occurrence de f est un préfixe du mot *aabaabaa* alors que la seconde occurrence en est un suffixe.

Dans tout l'exercice, Σ désigne un alphabet et f désigne un mot de longueur non nulle sur Σ.

❒ 1 – Montrer que le langage L_1 sur Σ des mots qui contiennent au moins une occurrence du facteur f est rationnel.

❒ 2 – Montrer que le langage L_2 sur Σ des mots qui contiennent au moins deux occurrences disjointes du facteur f est rationnel.

❒ 3 – Montrer que le langage L_3 sur Σ des mots qui contiennent au moins deux occurrences non disjointes du facteur f est rationnel.

❒ 4 – Montrer que le langage L_4 sur Σ des mots qui contiennent exactement une occurrence du facteur f est rationnel.

Problème d'algorithmique et programmation

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 d'épreuve 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 cela est nécessaire. Lorsque le candidat écrira une fonction ou une procédure, il pourra faire appel à une autre fonction ou procédure définie dans les questions précédentes. Enfin, si les paramètres d'une fonction ou d'une procédure à écrire sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction ou de cette procédure de tester si les hypothèses sont bien vérifiées.
Dans les énoncés du problème, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain (par exemple n).

Un *graphe* G est défini par deux ensembles X et E. L'ensemble X est un ensemble fini d'éléments appelés *sommets*. L'ensemble E est un ensemble de paires de sommets. Un élément $\{x, y\}$ de E est appelé *arête* de G ; x et y sont les *extrémités* de l'arête. L'*ordre* d'un graphe G est le nombre de sommets de G. Un graphe est représenté par un dessin où des cercles représentent les sommets et un trait joignant deux cercles représente une arête composée des deux sommets correspondant aux cercles. Si $\{x, y\}$ est une arête de G, on dit que x et y sont *voisins*. Le *degré* d'un sommet x est le nombre de voisins de x.
On dit que deux arêtes d'un graphe G sont *incidentes* si elles ont une extrémité en commun. On appelle *couplage dans* G un ensemble d'arêtes de G deux à deux non incidentes.
Un graphe G est dit *biparti* si on peut partitionner son ensemble de sommets X en deux sous-ensembles A et B ($A \neq \emptyset$, $B \neq \emptyset$, $A \cup B = X$, $A \cap B = \emptyset$) de sorte que toute arête ait une extrémité dans A et une extrémité dans B. Si les ensembles A et B ont même cardinal, on dit qu'il s'agit d'un *graphe biparti équilibré*. Dans tout le problème, on ne considère que des graphes bipartis équilibrés. On note n le cardinal commun aux ensembles A et B ; l'ordre du graphe est donc égal à $2n$. On suppose que l'on a toujours $n \geq 1$. Les sommets de A sont numérotés de 0 à $n - 1$ et nommés 0_A, 1_A, 2_A, …, $(n-1)_A$; les sommets de B sont numérotés de 0 à $n - 1$ et nommés 0_B, 1_B, 2_B, …, $(n-1)_B$. Une arête de G est toujours écrite en mettant d'abord l'extrémité qui est dans A puis celle qui est dans B.

On représente les graphes bipartis équilibrés par des schémas comme on peut le voir dans la figure 1 avec le graphe G_0, en représentant les sommets de A à gauche et les sommets de B à droite.

Pour G_0, n vaut 4 et l'ordre de G_0 vaut 8.
Les arêtes de G_0 sont :
$\{0_A, 0_B\}$, $\{0_A, 1_B\}$, $\{0_A, 2_B\}$, $\{1_A, 3_B\}$, $\{2_A, 0_B\}$, $\{2_A, 1_B\}$, $\{2_A, 2_B\}$, $\{2_A, 3_B\}$, $\{3_A, 3_B\}$.
Le sommet 0_A est de degré 3.
Le sommet 1_A est de degré 1.
Le sommet 2_A est de degré 4.
Le sommet 3_A est de degré 1.
Les sommets 0_B, 1_B et 2_B sont de degré 2.
Le sommet 3_B est de degré 3.

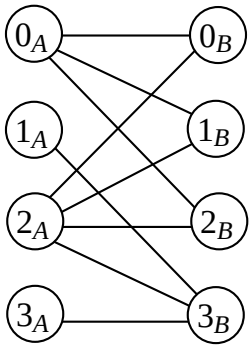

Figure 1 : le graphe G_0

Dans le graphe G_0, les arêtes $\{0_A, 0_B\}$ et $\{2_A, 3_B\}$ étant non incidentes, elles forment un couplage, nommé C_0, dont les arêtes sont dessinées en gras ci-contre ; on dit alors que dans ce couplage :

- le sommet 0_A est couplé au sommet 0_B, et réciproquement ;
- le sommet 2_A est couplé au sommet 3_B, et réciproquement ;
- les sommets 1_A, 3_A, 1_B et 2_B sont non couplés.

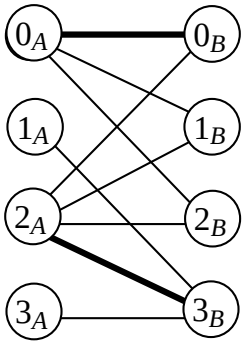

Figure 2 : le graphe G_0 et le couplage C_0

Le cardinal d'un couplage est le nombre d'arêtes de celui-ci ; par exemple le cardinal de C_0 vaut 2.

Première partie : généralités

❒ 5 – Exhiber un couplage de cardinal 3 dans G_0.

❒ 6 – Indiquer s'il existe dans G_0 un couplage de cardinal 4. Justifier la réponse.

Un graphe biparti équilibré d'ordre $2n$ est représenté par une matrice carrée de dimension $n \times n$ dont les lignes correspondent aux éléments de A et les colonnes aux éléments de B. Les cases de cette matrice sont indicées par (i, j) avec $0 \leq i \leq n-1$, $0 \leq j \leq n-1$ et contiennent des **valeurs booléennes** : la case d'indice (i, j) contient la valeur « vrai » (ou `true` dans le langage de programmation) si $\{i_A, j_B\}$ est une arête du graphe ; elle contient la valeur « faux » (ou `false` dans le langage de programmation) dans le cas contraire. Le graphe G_0 ci-dessus est donc représenté par la matrice suivante :

i \ j	0	1	2	3
0	vrai	vrai	vrai	faux
1	faux	faux	faux	vrai
2	vrai	vrai	vrai	vrai
3	faux	faux	faux	vrai

Figure 3 : la matrice représentant G_0

Un couplage est représenté par un tableau d'entiers indicé de 0 à $n - 1$. Soit i vérifiant $0 \leq i \leq n - 1$; si le sommet i_A est couplé avec le sommet j_B, la case d'indice i contient la valeur j ; si le sommet i_A n'est pas couplé, la case d'indice i contient une valeur égale à –1. Le couplage C_0 de G_0, formé des arêtes $\{0_A, 0_B\}$ et $\{2_A, 3_B\}$, est représenté par le tableau ci-dessous :

i	0	1	2	3
	0	–1	3	–1

Figure 4 : le tableau représentant C_0

Indications pour la programmation
Caml : Un vecteur est par la suite nommé aussi tableau.
On nomme *matrice* un tableau à deux dimensions (un vecteur de vecteurs).
La matrice représentant un graphe biparti équilibré d'ordre 2*n* est codée en Caml par une matrice $n \times n$ de booléens. La matrice représentant G_0 est ainsi codée par :

```
let G0 = [|[|true; true; true; false|];
          [|false; false; false; true|];
          [|true; true; true; true|];
          [|false; false; false; true|];|];;
```

Un couplage est codé par un vecteur de *n* entiers ; le couplage C_0 est ainsi codé par :

```
let C0 = [|0; -1; 3; -1|];;
```

Une arête *a* est codée par un vecteur `a` de deux entiers, avec `a.(0)` dans *A* et `a.(1)` dans *B*.
Fin des indications pour Caml

Pascal : on utilise les définitions suivantes :

```
const MAX = 100;

type Matrice = array[0 .. MAX - 1, 0 .. MAX - 1] of Boolean;
type Tableau = array[0 .. MAX - 1] of Integer;
type Arete = array[0 .. 1] of Integer;
```

La constante `MAX` donne une borne supérieure du cardinal des parties *A* et *B* des graphes bipartis équilibrés considérés, c'est-à-dire de la moitié de l'ordre du graphe ; il ne sera pas utile de vérifier cette condition dans la programmation.
Le type `Matrice` sert à coder les graphes bipartis équilibrés. La matrice représentant le graphe G_0 est ainsi codée par `G0` de type `Matrice` avec :

```
G0[0,0]:= true;  G0[0,1]:= true;  G0[0,2]:= true;  G0[0,3]:= false;
G0[1,0]:= false; G0[1,1]:= false; G0[1,2]:= false; G0[1,3]:= true;
G0[2,0]:= true;  G0[2,1]:= true;  G0[2,2]:= true;  G0[2,3]:= true;
G0[3,0]:= false; G0[3,1]:= false; G0[3,2]:= false; G0[3,3]:= true;
```

Le type `Tableau` a plusieurs usages. Il sert entre autres à coder un couplage ; le couplage C_0 est ainsi codé par `C0` de type `Tableau` avec :

```
C0[0] := 0; C0[1] := -1; C0[2] := 3; C0[3] := -1;
```

Une arête *a* est codée par un tableau `a` de type `Arete`, avec `a[0]` dans *A* et `a[1]` dans *B*.

Fin des indications pour Pascal

❒ 7 – Soit *G* un graphe biparti équilibré d'ordre 2*n*. On considère un tableau *C* d'entiers de longueur *n* et contenant dans ses cases indicées de 0 à *n* – 1 soit la valeur –1, soit une valeur comprise entre 0 et *n* – 1. Il s'agit de savoir si ce tableau *C* représente ou non un couplage dans *G*.

Caml : Écrire en Caml une fonction `verifie` telle que,

- si `G` est une matrice codant le graphe *G*,
- si `C` est un vecteur codant le tableau *C*,

alors `verifie G C` renvoie `true` si le tableau *C* représente un couplage dans *G* et `false` sinon.

Indiquer la complexité de la fonction `verifie`.

Pascal : Écrire en Pascal une fonction `verifie` telle que,

- si `G`, de type `Matrice`, code le graphe *G*,
- si `C`, de type `Tableau`, code le tableau *C*,
- si `n`, de type `Integer`, contient la valeur de *n*,

alors `verifie(G, C, n)` renvoie `true` si le tableau *C* représente un couplage dans *G* et `false` sinon.

Indiquer la complexité de la fonction `verifie`.

❒ 8 – On considère un tableau *C*, de longueur *n*, codant un couplage d'un graphe *G*. Il s'agit d'écrire une fonction qui calcule le cardinal de ce couplage.

Caml : Écrire en Caml une fonction `cardinal` telle que, si `C` est un vecteur codant un couplage, alors `cardinal C` renvoie le cardinal de ce couplage.

Indiquer la complexité de la fonction `cardinal`.

Pascal : Écrire en Pascal une fonction `cardinal` telle que,

- si `C`, de type `Tableau`, code un couplage *C*,
- si `n`, de type `Integer`, contient la valeur de *n*,

alors `cardinal(C, n)` renvoie le cardinal de *C*.

Indiquer la complexité de la fonction `cardinal`.

Deuxième partie : un algorithme pour déterminer un couplage maximal

On dit qu'un couplage C dans un graphe G est *maximal* si toute arête de G n'appartenant pas à C est incidente à au moins une arête de C. Par exemple, le couplage C_0 de G_0 est maximal. Un couplage maximal de G n'est pas forcément de cardinal maximum parmi les couplages de G. On cherche à concevoir un algorithme qui détermine un couplage maximal dans un graphe biparti équilibré G.

L'algorithme, nommé *algo_approche*, est le suivant :

- on commence avec un couplage vide C ;
- tant que G possède au moins une arête :
  - on choisit une arête a de G dont la somme des degrés des extrémités soit minimum ;
  - on ajoute l'arête a au couplage C ;
  - on retire de G l'arête a et toutes les arêtes incidentes à a.

On admettra que le résultat est, par construction, un couplage maximal.

❒ 9 – Appliquer *algo_approche* au graphe G_0 (voir la figure 1 page 4).

On considère par la suite le graphe biparti équilibré G_1 d'ordre 12 représenté sur la figure 5.

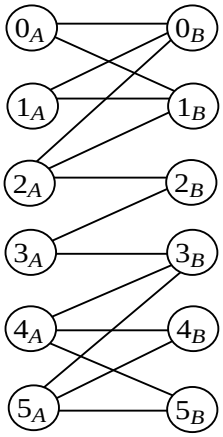

Figure 5 : le graphe G_1

❒ 10 – On applique *algo_approche* au graphe G_1. Déterminer la première arête a_1 choisie par *algo_approche* ; tracer le graphe obtenu après suppression de a_1 et des arêtes incidentes à a_1. Montrer que le couplage obtenu par *algo_approche* est de cardinal au plus 5 et indiquer s'il est de cardinal maximum parmi les couplages de G_1.

❒ 11 – Soit G un graphe biparti équilibré d'ordre $2n$. Il s'agit d'écrire en langage de programmation une fonction *arete_min* qui détermine une arête de G dont la somme des degrés des extrémités soit minimum. Si le graphe possède au moins une arête, cette fonction modifie un tableau a de deux entiers reçu en paramètre pour mettre dans les deux cases de a les numéros des deux extrémités d'une arête qui atteint ce minimum ; dans ce cas, la fonction renvoie la valeur « vrai » (`true`) ; sinon, elle renvoie la valeur « faux » (`false`).

Caml : Écrire en Caml une fonction `arete_min` telle que :

- si `G` est une matrice codant le graphe G,
- si `a` est un vecteur de deux entiers,

alors `arete_min G a` effectue les opérations décrites ci-dessus, en modifiant le tableau `a` dans le cas où G possède au moins une arête.

Indiquer la complexité de la fonction `arete_min`.

Pascal : Écrire en Pascal une fonction `arete_min` telle que :
- si `G`, de type `Matrice`, code le graphe *G*,
- si `n`, de type `Integer`, contient la valeur de *n*,
- si `a` est de type `Arete`,

alors `arete_min(G, n, a)` effectue les opérations décrites ci-dessus en modifiant le tableau `a` dans le cas où *G* possède au moins une arête.
Indiquer la complexité de la fonction `arete_min`.

❒ 12 – Il s'agit d'écrire en langage de programmation une fonction ou une procédure *supprimer* qui supprime d'un graphe biparti équilibré une arête *a* donnée et toutes les arêtes incidentes à *a*.
Caml : Écrire en Caml une fonction `supprimer` telle que :
- si `G` est une matrice codant un graphe biparti équilibré *G*,
- si `a` est un vecteur de deux entiers codant une arête *a* de *G*,

alors `supprimer G a` modifie `G` pour que, après modifications, `G` code le graphe obtenu à partir de *G* en supprimant *a* et toutes les arêtes incidentes à *a*.
Indiquer la complexité de la fonction `supprimer`.

Pascal : Écrire en Pascal une procédure `supprimer` telle que :
- si `G`, de type `Matrice`, code un graphe biparti équilibré *G* d'ordre 2*n*,
- si `n`, de type `Integer`, contient la valeur de *n*,
- si `a`, de type `Arete`, code une arête *a* de *G*,

alors `supprimer(G, n, a)` modifie `G` pour que, après l'appel à la procédure, `G` code le graphe obtenu à partir de *G* en supprimant *a* et toutes les arêtes incidentes à *a*.
Indiquer la complexité de la procédure `supprimer`.

❒ 13 – Il s'agit de définir en langage de programmation l'algorithme *algo_approche* décrit au début de la deuxième partie.
Caml : Écrire en Caml une fonction `algo_approche` telle que, si `G` est une matrice qui code un graphe biparti équilibré *G*, `algo_approche G` effectue *algo_approche* à partir d'une copie de `G` et renvoie un vecteur codant le couplage obtenu.
Indication : on pourra utiliser sans la définir une fonction `dupliquer_matrice` telle que, si `G` est une matrice codant un graphe biparti équilibré *G*, `dupliquer_matrice G` renvoie une matrice identique à `G`. Cette fonction sera utilisée pour que la fonction `algo_approche` ne modifie pas le contenu de la matrice reçue en paramètre.
Indiquer la complexité de la fonction `algo_approche`.

Pascal : Écrire en Pascal une fonction `algo_approche` telle que :
- si `G`, de type `Matrice`, code un graphe biparti équilibré *G* d'ordre 2*n*,
- si `n`, de type `Integer`, contient la valeur de *n*,

alors `algo_approche(G, n)` effectue *algo_approche* et renvoie un tableau de type `Tableau` codant le couplage obtenu. On fera en sorte que la fonction ne modifie pas la matrice `G`.
Indiquer la complexité de la fonction `algo_approche`.

Troisième partie : recherche exhaustive d'un couplage de cardinal maximum

❒ 14 – Soit *G* un graphe biparti équilibré d'ordre 2*n*. Il s'agit d'écrire en langage de programmation une fonction nommée *une_arete* qui recherche une arête quelconque de *G*. Si *G* possède au moins une arête, la fonction mémorise **la première arête rencontrée** *a* dans un tableau (ou vecteur en Caml) de deux entiers reçu en paramètre ; elle arrête alors sa recherche et renvoie la valeur « vrai » (`true`) ; sinon, la fonction renvoie la valeur « faux » (`false`).

Caml : Écrire en Caml une fonction `une_arete` telle que :

- si `G` est une matrice codant le graphe *G*,
- si `a` est un vecteur de deux entiers destiné à coder l'arête *a*,

alors `une_arete G a` effectue les opérations décrites ci-dessus, en modifiant le vecteur `a` dans le cas où *G* possède au moins une arête.

Pascal : Écrire en Pascal une fonction `une_arete` telle que :

- si `G`, de type `Matrice`, code le graphe *G*,
- si `n`, de type `Integer`, contient la valeur de *n*,
- si `a`, de type `Arete`, est destiné à coder l'arête *a*,

alors `une_arete(G, n, a)` effectue les opérations décrites ci-dessus en modifiant le tableau `a` dans le cas où *G* possède au moins une arête.

❒ 15 – On cherche à établir un algorithme **récursif**, nommé *meilleur_couplage*, qui permette de déterminer un couplage de cardinal maximum dans un graphe biparti équilibré. Le principe est le suivant.

Si le graphe courant ne contient aucune arête, le cardinal maximum d'un couplage est 0 et aucun sommet n'est couplé.

Dans le cas contraire, l'algorithme considère une arête quelconque *a* du graphe courant et recherche successivement :

- un couplage de cardinal maximum parmi les couplages du graphe courant ne contenant pas *a*
- un couplage de cardinal maximum parmi les couplages du graphe courant contenant *a.*

L'algorithme déduit alors un couplage de cardinal maximum.

Caml : Écrire en Caml une fonction récursive `meilleur_couplage` telle que, si `G` est une matrice codant un graphe biparti équilibré *G*, `meilleur_couplage G` renvoie un vecteur codant un couplage de cardinal maximum dans *G*. La fonction utilisera le principe décrit plus haut.

Indication : on pourra utiliser sans la définir une fonction `dupliquer_matrice` telle que, si `G` est une matrice codant un graphe biparti équilibré *G*, alors `dupliquer_matrice G` renvoie une matrice identique à `G`.

Pascal : Écrire en Pascal une fonction récursive `meilleur_couplage` telle que :

- si `G`, de type `Matrice`, code un graphe biparti équilibré *G* d'ordre 2*n*,
- si `n`, de type `Integer`, contient la valeur de *n*,

alors `meilleur_couplage(G, n)` renvoie un tableau de type `Tableau` codant un couplage de cardinal maximum dans *G*. La fonction utilisera le principe décrit plus haut.

Quatrième partie : l'algorithme hongrois

On considère un graphe biparti équilibré G et un couplage C. Une *chaîne* de G est une suite $x_0, x_1, \ldots, x_k, \ldots, x_p$ ($p \geq 0$) de sommets **distincts** telle que, pour k compris entre 0 et $p - 1$, $\{x_k, x_{k+1}\}$ est une arête de G. Le sommet x_0 s'appelle l'*origine* de la chaîne et le sommet x_p l'*extrémité* de la chaîne.

Une chaîne $x_0, x_1, \ldots, x_k, \ldots, x_p$ de G, avec $p \geq 0$, est dite *alternée relativement à C* si :

- pour tout indice k pair, x_k est dans A,
- pour tout indice k impair, x_k est dans B,
- le sommet x_0 n'est pas couplé,
- pour tout entier i vérifiant $0 \leq 2i \leq p - 1$, l'arête $\{x_{2i}, x_{2i+1}\}$ n'appartient pas à C,
- pour tout entier i vérifiant $2 \leq 2i \leq p$, l'arête $\{x_{2i-1}, x_{2i}\}$ appartient à C.

Autrement dit :

- l'origine de la chaîne est dans A et n'est pas couplée,
- la première arête de la chaîne n'est pas dans C, la deuxième est dans C, la troisième n'est pas dans C et ainsi de suite.

Une chaîne $x_0, x_1, \ldots, x_p$ alternée relativement au couplage C est dite *chaîne alternée augmentante relativement à C* si on a $p \geq 1$ et si de plus x_p n'est pas couplé, ce qui entraîne que x_p est dans B. Par exemple, dire qu'une chaîne $x_0, x_1, x_2, x_3, x_4, x_5$ constitue une chaîne alternée augmentante relativement à un couplage C signifie que :

- x_0, x_2 et x_4 sont des sommets de A, x_1, x_3 et x_5 sont des sommets de B ;
- $\{x_0, x_1\}, \{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}$ sont des arêtes de G ;
- x_0 n'est pas couplé dans C, x_5 n'est pas couplé dans C ;
- x_1 est couplé avec x_2, x_2 n'est pas couplé avec x_3 et x_3 est couplé avec x_4.

❒ 16 – On considère le graphe G_1 et le couplage C_1 constitué des arêtes $\{0_A, 0_B\}, \{1_A, 1_B\}, \{3_A, 2_B\}, \{4_A, 3_B\}, \{5_A, 5_B\}$ représentées sur la figure 6 en gras. Après avoir indiqué le seul sommet de A qui puisse être l'origine d'une chaîne alternée augmentante relativement à C_1 et le seul sommet de B qui puisse être l'extrémité d'une chaîne alternée augmentante relativement à C_1, déterminer une chaîne alternée augmentante relativement à C_1.

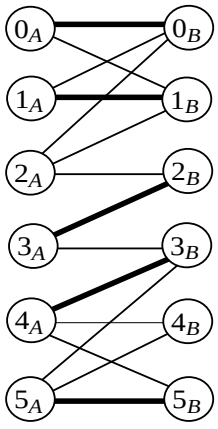

Figure 6 : le graphe G_1 et le couplage C_1

❒ 17 – On considère un graphe biparti équilibré G et un couplage C dans G. Montrer que s'il existe une chaîne alternée augmentante relativement à C, alors il existe dans G un couplage dont le cardinal est égal au cardinal de C augmenté de 1.

On admet le théorème suivant : *un couplage C d'un graphe biparti équilibré G est de cardinal maximum si et seulement s'il n'existe pas dans G de chaîne alternée augmentante relativement à C.*

Soit *G* un graphe biparti équilibré. L'*algorithme hongrois* s'appuie sur le théorème ci-dessus et détermine un couplage de cardinal maximum dans *G*. L'algorithme débute avec un couplage *C* de cardinal nul ; tant qu'il existe une chaîne augmentante relativement à *C*, l'algorithme modifie *C* pour incrémenter de 1 le cardinal du couplage en utilisant une telle chaîne augmentante.
La suite du problème a pour objectif de programmer cet algorithme. On commence par étudier la recherche d'une chaîne alternée augmentante.

On considère un graphe biparti équilibré *G* et un couplage *C* dans *G*. On va chercher s'il existe une chaîne alternée augmentante relativement à *C*. Pour cela, on recherche les sommets qui sont extrémités de chaînes alternées en procédant de proche en proche à partir des sommets de *A* non couplés.
Un sommet *y* est dit *atteint* si une chaîne alternée relativement à *C* et d'extrémité *y* est mise en évidence. Au départ, tous les sommets non couplés de *A* (un tel sommet est extrémité d'une chaîne alternée réduite à ce sommet) sont considérés comme atteints ; aucun autre sommet n'est considéré comme atteint. Un sommet *y* **non encore atteint** peut être atteint à partir d'un de ses voisins *x* déjà atteint si on a :

- soit *y* est dans *B* (et donc *x* est dans *A*) et l'arête $\{x, y\}$ n'est pas dans le couplage *C* ;
- soit *y* est dans *A* (et donc *x* est dans *B*) et l'arête $\{y, x\}$ est dans le couplage *C*.

On utilise des *marques attribuées aux sommets*. Ces marques sont des entiers initialisés à –1 pour tous les sommets. Lorsqu'un sommet *y* est atteint à partir d'un sommet *x*, la marque de *y* devient égale au numéro de *x* (on rappelle que le *numéro* d'un sommet i_A ou d'un sommet i_B vaut *i*) ; *x* est alors l'avant-dernier sommet dans la chaîne alternée *Ch*(*y*) d'extrémité *y* mise en évidence. La chaîne *Ch*(*y*) peut être retrouvée à l'envers, de proche en proche, grâce aux marques ; dans cette chaîne, seule l'origine porte une marque de valeur –1.
Si un sommet non couplé *y* de *B* est atteint, *Ch*(*y*) est une chaîne alternée augmentante relativement à *C*.
Dans le cas où simultanément :

- il n'y a plus de sommet non encore atteint qui puisse être atteint,
- aucun sommet non couplé de *B* n'est atteint,

on admet qu'il n'existe pas de chaîne alternée augmentante relativement à *C*.
Remarque : les valeurs des marques peuvent dépendre de l'ordre dans lequel on atteint les sommets, sans que cela n'ait d'importance pour la suite du problème.

❒ 18 – On considère le graphe G_1 et un couplage nommé C_1' constitué des arêtes $\{0_A, 0_B\}$, $\{2_A, 1_B\}$, $\{3_A, 2_B\}$, $\{4_A, 4_B\}$, $\{5_A, 5_B\}$, en gras sur la figure 7.
Certains sommets ont été atteints ; sur la figure 7, les marques attribuées sont portées à côté des sommets, les sommets atteints sont encadrés.
Utiliser les marques pour reconstituer la chaîne alternée arrivant dans le sommet 3_B et correspondant aux marques. Indiquer s'il s'agit d'une chaîne alternée augmentante relativement à C_1'.

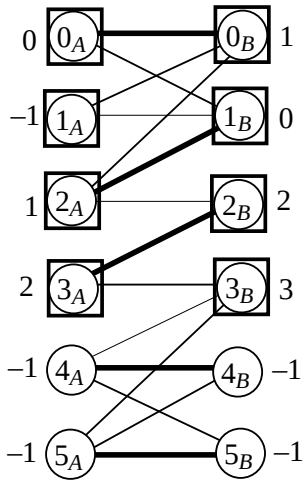

Figure 7 : le graphe G_1 et le couplage C_1'

On considère quatre tableaux :

- Un tableau nommé `C` codant un couplage *C*.
- Un tableau nommé `R` (pour réciproque) indicé par 0, 1, …, $n - 1$; soit j vérifiant $0 \leq j \leq n - 1$; si le sommet j_B n'est pas couplé dans *C*, la case d'indice j du tableau `R` contient la valeur –1 ; si le sommet j_B est couplé avec le sommet i_A, la case d'indice j du tableau `R` comporte la valeur i.
- Un tableau nommé `mA` indicé par 0, 1, …, $n - 1$ servant à contenir les marques des sommets de *A*.
- Un tableau nommé `mB` indicé par 0, 1, …, $n - 1$ servant à contenir les marques des sommets de *B*.

❒ 19 – On suppose qu'une chaîne $Ch(x_p) = x_0, x_1, \ldots, x_p$ alternée augmentante relativement à un couplage *C* a été déterminée et codée grâce aux marques. D'après la question ❒ 17, il existe un couplage de cardinal égal à celui de *C* augmenté de 1. La fonction ou procédure `actualiser` reçoit en paramètres les quatre tableaux décrits ci-dessus ainsi que le numéro de x_p ; elle transforme alors les tableaux `C` et `R` pour qu'ils correspondent à un couplage, obtenu à partir de *C* et de $Ch(x_p)$, dont le cardinal est celui du couplage *C* augmenté de 1.

Caml : Écrire en Caml une fonction `actualiser` telle que :

- si `C`, `R`, `mA`, `mB` sont des vecteurs de n entiers qui correspondent à la description donnée plus haut,
- si `numero` est un entier donnant le numéro de x_p,

alors `actualiser C R mA mB numero` modifie les vecteurs `C` et `R` pour obtenir un couplage de cardinal égal à celui de *C* augmenté de 1.

Pascal : Écrire en Pascal une procédure `actualiser` telle que :

- si les tableaux `C`, `R`, `mA`, `mB`, de type `Tableau`, correspondent à la description donnée plus haut,
- si `numero` est un entier donnant le numéro de x_p,

alors `actualiser(C, R, mA, mB, numero)` modifie les tableaux `C` et `R` pour obtenir un couplage de cardinal égal à celui de *C* augmenté de 1.

❒ 20 – On considère le graphe G_2 ci-contre et un couplage, nommé C_2, constitué des arêtes $\{0_A, 0_B\}$, $\{2_A, 2_B\}$, $\{3_A, 3_B\}$, $\{4_A, 4_B\}$ en gras sur la figure 8. Recopier la figure, encadrer tous les sommets qui peuvent être atteints et préciser à côté des sommets les marques obtenues. Indiquer s'il existe dans G_2 une chaîne alternée augmentante relativement à C_2.

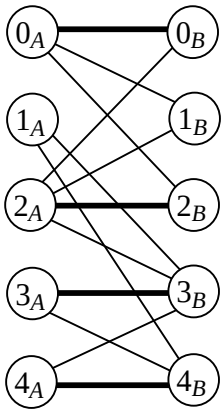

Figure 8 : le graphe G_2 et le couplage C_2

Indication : on procédera de proche en proche à partir du seul sommet non couplé de *A*, c'est-à-dire à partir du sommet 1_A. L'ordre dans lequel on atteint les sommets n'a pas d'importance.

On suppose donnés un graphe biparti équilibré *G* et un couplage *C* dans *G*. Les chaînes alternées considérées dans la suite sont toujours des chaînes alternées relativement à *C*. Les questions ❒ 21 et ❒ 22 ont pour objectif de programmer un algorithme qui cherche une chaîne alternée augmentante en atteignant les sommets de proche en proche à partir des sommets non couplés de *A*.

❒ 21 – On suppose que certains sommets de *G* peuvent déjà avoir été atteints et les marques calculées en conséquence. On note *x* un sommet de *A* ou de *B* déjà atteint.
On définit deux nouvelles fonctions, les fonctions *chercherA* et *chercherB*.
Appliquée au sommet *x*, appelé *sommet de* départ, la fonction *chercherA* (si *x* est dans *A*) ou la fonction *chercherB* (si *x* est dans *B*) détermine des sommets non encore atteints qui peuvent être atteints, **de voisin en voisin**, **récursivement**, à partir de *x*, modifie les marques de ces sommets nouvellement atteints et s'arrête dans les cas suivants :

- un sommet *y* de *B* non couplé a été atteint ; elle renvoie alors le numéro du sommet *y* ;
- tous les sommets qui peuvent être atteints de voisin en voisin à partir de *x* l'ont été et aucun sommet non couplé de *B* n'est atteint ; elle renvoie alors la valeur –1.

Il s'agit d'écrire les deux fonctions *chercherA* et *chercherB* en utilisant une récursivité croisée, chacune des deux fonctions pouvant faire appel à l'autre.
Caml : Définir ce qu'on appelle récursivité croisée et indiquer comment elle peut être implémentée en Caml.
Écrire en Caml les deux fonctions *chercherA* et *chercherB* ; chacune de ces deux fonctions reçoit en paramètres :

- une matrice `G` codant le graphe *G* ;
- quatre vecteurs de longueur *n* pour les quatre tableaux décrits plus haut : `C`, `R`, `mA` et `mB` ;
- un entier codant le numéro du sommet de départ de la recherche.

Ces deux fonctions modifient les vecteurs `mA` et `mB` conformément à la description ci-dessus. Elles renvoient le numéro d'un sommet atteint non couplé de *B* ou la valeur –1 selon le cas.

Pascal : Définir ce qu'on appelle récursivité croisée et indiquer comment elle peut être implémentée en Pascal.
Écrire en Pascal les deux fonctions *chercherA* et *chercherB* ; chacune de ces deux fonctions reçoit en paramètres :

- une matrice `G`, de type `Matrice`, codant le graphe *G* ;
- quatre tableaux de type `Tableau` pour les quatre tableaux décrits plus haut : `C`, `R`, `mA` et `mB` ;
- un entier `n` contenant la valeur de *n* donnant le cardinal commun à *A* et *B* ;
- un entier codant le numéro du sommet de départ de la recherche.

Ces deux fonctions modifient les tableaux `mA` et `mB` conformément à la description ci-dessus. Elles renvoient le numéro d'un sommet atteint non couplé de *B* ou la valeur –1 selon le cas.

❒ 22 – On suppose donnés un graphe biparti équilibré G d'ordre $2n$ et un couplage C dans G. Tous les sommets de G possèdent une marque égale à –1.
La fonction *chaine_alternee* cherche s'il existe une chaîne alternée augmentante en appliquant la fonction *chercherA* successivement à partir des sommets non couplés de A.
Caml : Écrire en Caml la fonction *chaine_alternee* telle que :

- si `G` est une matrice codant le graphe G,
- si `C`, `R`, `mA` et `mB` correspondent à la description donnée précédemment, toutes les cases de `mA` et `mB` étant initialisées à –1,

alors `chaine_alternee G C R mA mB` renvoie :

- –1 s'il n'existe pas de chaîne alternée augmentante,
- le numéro de l'extrémité d'une chaîne alternée augmentante dans le cas contraire.

De plus, la fonction modifie les vecteurs `mA` et `mB` pour qu'ils contiennent les marques des sommets à la fin de l'exécution de la fonction.

Pascal : Écrire en Pascal la fonction *chaine_alternee* telle que :

- si `G`, de type `Matrice`, code le graphe G,
- si `C`, `R`, `mA` et `mB`, de type `Tableau`, correspondent à la description donnée précédemment, les cases de `mA` et `mB` d'indices compris entre 0 et $n - 1$ étant initialisées à –1,
- si `n`, de type `Integer`, contient la valeur de n,

alors `chaine_alternee(G, C, R, mA, mB, n)` renvoie :

- –1 s'il n'existe pas de chaîne alternée augmentante,
- le numéro de l'extrémité d'une chaîne alternée augmentante dans le cas contraire.

De plus, la fonction modifie les tableaux `mA` et `mB` pour qu'ils contiennent les marques des sommets à la fin de l'exécution de la fonction.

❒ 23 – Dans cette question, on programme l'algorithme hongrois.
Caml : Écrire en Caml la fonction `algorithme_hongrois` telle que, si `G` est une matrice codant un graphe biparti équilibré G, alors `algorithme_hongrois G` renvoie un vecteur codant le couplage obtenu par l'algorithme hongrois.

Pascal : Écrire en Pascal la fonction `algorithme_hongrois` telle que :

- si `G`, de type `Matrice`, code un graphe biparti équilibré G d'ordre $2n$,
- si `n`, de type `Integer`, contient la valeur de n,

alors `algorithme_hongrois(G, n)` renvoie un tableau de type `Tableau` codant le couplage obtenu par l'algorithme hongrois.