Classements de tournois, coloration de graphes cordés

Ce devoir est constitué de deux partes totalement indépendantes :

- une première partie algorithmique qui peut être traitée, à votre choix, en C ou en OCaml (on rappelle qu'il est possible de définir des fonctions ou des procédures auxiliaires que l'on explicitera, ou de faire appel à d'autres fonctions ou procédures définies dans les questions précédentes et comme toujours, votre code doit être commenté et/ou expliqué) ;
- une seconde partie plus théorique où un soin particulier est attendu dans les preuves et justifications.

1 Ordres pour un tournoi

1.1 Tournois

Dans ce problème, on note `true` et `false` les deux valeurs possibles d'une variable booléenne. On considérera des matrices carrées ; les lignes et colonnes d'une matrice de dimensions $n \times n$ seront toujours numérotées de 0 à $n-1$. Si T est une matrice de taille $n \times n$, pour $0 \leqslant i < n$ et $0 \leqslant j < n$, $t_{i,j}$ représentera le coefficient de T situé sur la ligne i et la colonne j.

On appelle *tournoi* une matrice carrée T à coefficients booléens qui, si la matrice est de dimensions $n \times n$, vérifie :

- pour $0 \leqslant i < n$ et $0 \leqslant j < n$ avec $i \neq j$, $t_{i,j} =$ `true` $\iff t_{j,i} =$ `false` ;
- pour $0 \leqslant i < n$, $t_{i,i} =$ `false`.

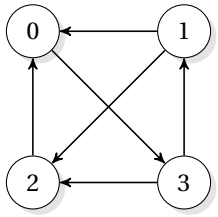
Si la matrice est de taille $n \times n$, le tournoi est dit d'ordre n. L'ordre n des tournois considérés dans ce problème sera toujours au moins égal à 1.

On associe un tournoi T à un graphe orienté G de la façon suivante : à chaque entier i vérifiant $0 \leqslant i < n$, on associe un sommet i ; pour tout couple d'entiers i et j vérifiant $0 \leqslant i < n$ et $0 \leqslant j < n$ et $i \neq j$, si $t_{i,j}$ vaut `true`, on ajoute à ce graphe un arc de i vers j.

En d'autres termes, le tournoi correspond à la matrice d'adjacence du graphe G !

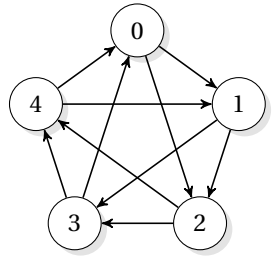
Le tournoi T_4 défini ci-dessous à gauche est représenté par le graphe G_4 qui se trouve à sa droite :

```
[false  false  false  true ]
[true   false  true   false]
[true   false  false  false]
[false  true   true   false]
```



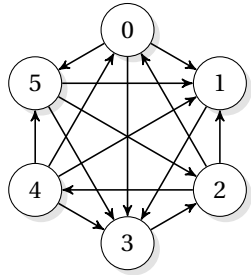
On utilisera aussi le tournoi T_5 défini à gauche ci-dessous et représenté par le graphe G_5 qui se trouve à sa droite :

```
[false  true   true   false  false]
[false  false  true   true   false]
[false  false  false  true   true ]
[true   false  false  false  true ]
[true   true   false  false  false]
```



On utilisera enfin le tournoi T_6 défini à gauche ci-dessous et représenté par le graphe G_6 qui se trouve à sa droite :

```
[false  true   false  true   false  true ]
[false  false  false  true   false  false]
[true   true   false  false  true   false]
[false  false  true   false  false  false]
[true   true   false  true   false  true ]
[false  true   true   true   false  false]
```



1. Combien le graphe associé à un tournoi d'ordre n possède-t-il d'arcs ?

En OCaml, un tournoi sera représenté par une matrice de booléens (`bool array array`). En langage C, un tournoi sera représenté par un objet de type `bool** t`. Pour un tel objet `t`, on peut accéder au booléen ligne i colonne j en écrivant `t[i][j]`. Il n'est pas besoin de savoir construire de tels objets, ils seront fournis à des fonctions qui n'auront qu'à les lire.

On s'intéresse à un jeu nommé J qui se joue à deux joueurs. Pour chaque partie du jeu J, il y a un gagnant et un perdant, il n'y a pas de match nul. Soit n un entier strictement positif. On considère un ensemble de n joueurs. Une *compétition* C du jeu J effectuée par les n joueurs consiste à ce que chaque joueur joue une et une seule fois au jeu J contre chaque autre joueur.

Les joueurs sont identifiés par des numéros allant de 0 à $n-1$. Le résultat de cet ensemble de $n(n-1)/2$ parties est représenté par un tournoi T d'ordre n : pour i et j distincts vérifiant $0 \leqslant i < n$ et $0 \leqslant j < n$, le coefficient $t_{i,j}$ de T vaut `true` si le joueur i a gagné contre le joueur j et `false` sinon. Pour $0 \leqslant i < n$, le coefficient $t_{i,i}$ vaut `false`. On dira que *le tournoi* T *représente le résultat de la compétition* C.

Par exemple, pour une compétition à 4 joueurs représentée par le tournoi T_4 ci-dessus :

- le joueur 0 a gagné contre le joueur 3 et perdu contre les joueurs 1 et 2,
- le joueur 1 a gagné contre les joueur 0 et 2 et perdu contre le joueurs 3,
- le joueur 2 a gagné contre le joueur 0 et perdu contre les joueurs 1 et 3,
- le joueur 3 a gagné contre les joueur 1 et 2 et perdu contre le joueurs 0.

On appelle *classement d'ordre n* toute permutation des entiers $[\![0\,..\,n-1]\!]$. Un classement σ d'ordre n sera noté $(\sigma(0),\sigma(1),\ldots,\sigma(n-1))$. Après une compétition entre n joueurs, un classement σ d'ordre n est interprété comme un classement des joueurs par résultats décroissants. Le joueur $\sigma(0)$ est considéré comme étant le meilleur tandis que le joueur $\sigma(n-1)$ est considéré comme étant le moins bon. Un joueur a est mieux placé qu'un joueur b si le joueur a apparaît avant le joueur b dans le classement. Par exemple, pour quatre joueurs, le classement $(1,3,2,0)$ est interprété comme : 1 est meilleur que 3, qui est meilleur que 2, qui est meilleur que 0.

En OCaml comme en langage C, un classement sera représenté par un tableau d'entiers (un `int array` en OCaml, un pointeur vers un tableau d'entiers en langage C).

Après une compétition, on peut chercher à déterminer le classement qui représente le mieux le résultat de la compétition. Il y a différentes méthodes permettant de faire cela, ces méthodes ne donnent pas toutes le même classement. On étudie deux d'entre elles dans ce problème, la *méthode de Copeland* et la *méthode de Slater*.

1.2 Méthode de Copeland

Dans un tournoi T, on appelle *score* de i $(0 \leqslant i < n)$ le nombre de termes égaux à `true` sur la ligne i de T ; par rapport à la compétition du jeu J décrite ci-dessus, le score de i est le nombre de parties gagnées par le joueur i. Par exemple, si la compétition est décrite par T_4, les scores des joueurs 0, 1, 2 et 3 sont respectivement égaux à 1, 2, 1 et 2.

Soit T un tournoi représentant une compétition C, un classement est appelé *classement de Copeland* pour le tournoi T si les scores des joueurs dans ce classement sont décroissants au sens large. Le classement $(1,3,0,2)$ est un classement de Copeland pour le tournoi T_4 puisque, dans l'ordre de ce classement, les scores forment la suite décroissante 2, 2, 1 et 1.

2. Indiquer un classement de Copeland pour le tournoi T_6.

3. Écrire une fonction `calc_scores` telle que, si `t` est une matrice de booléens codant un tournoi T d'ordre n, alors `calc_scores t` (en OCaml) ou `calc_scores(t, n)` (en langage C) retourne un tableau d'entiers (`int Array` en OCaml, un pointeur `int*` vers un tableau d'entiers alloué par la fonction en C) de longueur n contenant, pour i qui varie de 0 à $n-1$, le score du joueur i à l'indice i. Préciser la complexité de cette fonction.

```
val calc_scores : bool array array -> int array = <fun>
```

```
int* calc_scores(bool** t, int n);
```

4. Écrire une fonction `order_Copeland` telle que, si `t` est une matrice de booléens codant un tournoi T d'ordre n, alors `order_Copeland t` (en OCaml) ou `order_Copeland(t, n)` (en langage C) retourne un tableau d'entiers (`int Array` en OCaml, un pointeur `int*` vers un tableau d'entiers alloué par la fonction en C) de longueur n contenant un classement de Copeland pour T. On ne cherchera pas nécessairement la meilleure complexité, mais on précisera la complexité de la fonction proposée.

```
val order_Copeland : bool array array -> int array = <fun>
```

```
int* order_Copeland(bool** t, int n);
```

1.3 Valeur de Slater d'un classement

Soit $T = (t_{i,j})_{0\leqslant i<n, 0\leqslant j<n}$ un tournoi d'ordre n représentant le résultat d'une compétition C, et σ un classement d'ordre n. On dit qu'une partie jouée entre deux joueurs i et j pendant la compétition C est *contredite* par le classement σ si i se trouve avant j dans σ alors que i a perdu la partie contre j, ou bien si j est avant i dans σ alors que j a perdu la partie contre i. On appelle *valeur de Slater du classement* σ *pour* T, et on note $\text{Slater}(T,\sigma)$ le nombre de parties de C contredites par σ. Autrement dit, $\text{Slater}(T,\sigma)$ est le nombre de couples (i,j) vérifiant simultanément :

- $0 \leqslant i < j < n$,
- $t_{\sigma(i),\sigma(j)} =$ `false`.

À titre d'exemple, considérons le classement $\sigma_4 = (1,3,2,0)$; pour calculer $\text{Slater}(T_4,\sigma_4)$, on peut examiner successivement les valeurs de $t_{1,3}$ (qui vaut `false` et contribue donc pour 1), de $t_{1,2}$ (qui vaut `true`), de $t_{1,0}$ (qui vaut `true`), de $t_{3,2}$ (qui vaut `true`), de $t_{3,0}$ (qui vaut `false` et contribue donc pour 1), de $t_{2,0}$ (qui vaut `true`) : $\text{Slater}(T_4,\sigma_4)$ vaut donc 2, les parties contredites par le classement étant la partie entre 1 et 3 et la partie entre 0 et 3.

5. En posant $\sigma_5 = (2,4,0,1,3)$, calculer $\text{Slater}(T_5,\sigma_5)$.

6. En notant σ_6 le classement de Copeland déterminé à la question 1, calculer $\text{Slater}(T_6,\sigma_6)$.

7. Écrire une fonction `val_Slater` telle que, si `t` est une matrice de booléens codant un tournoi T d'ordre n, si `sigma` est un tableau représentant un classement σ d'ordre n, alors `val_Slater t sigma` (en OCaml) ou `val_Slater(t, n, sigma)` (en langage C) retourne $\text{Slater}(T,\sigma)$.

```
val val_Slater : bool array array -> int array -> int = <fun>
```

```
int val_Slater(bool** t, int n, int sigma[]);
```

8. Indiquer, en la justifiant, la complexité de la fonction `val_Slater`.

1.4 Indice de Slater d'un tournoi

On appelle *indice de Slater* d'un tournoi T d'ordre n et on note $s(T)$ le minimum de Slater(T,σ) sur l'ensembles des classements σ d'ordre n.

Un *classement de Slater* pour un tournoi T est un classement σ d'ordre n vérifiant Slater$(T,\sigma) = s(T)$. Ainsi, un classement de Slater pour T contredit un minimum de parties de la compétition représentée par le tournoi T.

9. Calculer $s(T_4)$ et indiquer un classement de Slater pour T_4.

Soit T un tournoi d'ordre n. Soient i et j deux entiers distincts compris entre 0 et $n-1$. On dit que (i,j) est *un arc* de T si $t_{i,j}$ est vrai. Cela signifie aussi que le joueur i a gagné contre le joueur j dans la compétition représentée par T, ou bien encore qu'il y a une flèche de i à j dans le graphe illustrant T.

Soit $p \geqslant 3$. On considère p entiers distincts $i_1, i_2, \ldots, i_p$ compris entre 0 et $n-1$. On dit que $(i_1, i_2, \ldots, i_p)$ est *un cycle* de T si $(i_1,i_2),(i_2,i_3),\ldots,(i_{p-1},i_p),(i_p,i_1)$ sont des arcs de T (autrement dit, i_1 a gagné contre i_2, i_2 a gagné contre i_3, ..., i_{p-1} a gagné contre i_p et enfin i_p a gagné contre i_1). On dira que $(i_1,i_2),(i_2,i_3),\ldots,(i_{p-1},i_p),(i_p,i_1)$ sont les arcs de ce cycle.

Si p vaut 3, on dit qu'il s'agit d'un *triangle* de T. On dit que deux cycles sont *arcs-disjoints* s'ils n'ont pas d'arc en commun. Par exemple, dans T_5, les cycles $(0,1,3)$ et $(0,2,3,4)$ sont arcs-disjoints.

10. On suppose qu'il existe m cycles deux à deux arc-disjoints dans le tournoi T. Indiquer en fonction de m un minorant de l'indice de Slater de T. Justifier la réponse.

11. On considère un tournoi T d'ordre n, un classement σ d'ordre n et un ensemble de m cycles deux à deux arc-distincts. On suppose que l'on a Slater$(T,\sigma) = m$. Indiquer alors ce qu'on peut dire de l'indice de Slater de T du classement σ.

12. Calculer $s(T_6)$ et indiquer un classement de Slater pour T_6. Déduire de ce résultat qu'un classement de Copeland pour un tournoi T n'est pas toujours un classement de Slater pour T.

On considère un ensemble E de triangles deux à deux arc-disjoints d'un tournoi T. On dit que cet ensemble est un ensemble *maximal de triangles deux à deux arc-disjoints* de T si tout triangle de T possède au moins un arc en commun avec au moins un triangle de E. On remarquera que l'ensemble E n'est pas nécessairement de cardinal maximum parmi les ensembles E de triangles deux à deux arc-disjoints du tournoi T.

13. Indiquer un ensemble maximal de triangles deux à deux arc-disjoints de T_5.

On souhaite calculer le cardinal d'un ensemble maximal de triangles deux à deux arc-disjoints d'un tournoi T d'ordre n grâce à une fonction `compter_triangles`. Pour cela, la fonction construit une suite de triangles deux à deux arc-disjoints (suite dont il n'est pas nécessaire de mémoriser les éléments) jusqu'à avoir un ensemble maximal.

14. Écrire une fonction `count_triangles` telle que si `t` est une matrice de booléens codant un tournoi T d'ordre n, alors `count_triangles t` (en OCaml) ou `count_triangles(bool** t, int n)` (en langage C) retourne le cardinal d'un ensemble maximal de triangles deux à deux arc-disjoints de T. La complexité de cette fonction devra être de l'ordre de n^3, mais on ne demande pas la justification.

```
val count_triangles : bool array array -> int = <fun>
```

```
int count_triangles(bool** t, int n);
```

1.5 Méthode de Slater

On se propose d'écrire une fonction qui, étant donné un tournoi T d'ordre n, détermine un classement de Slater pour T. Pour cela, on énumère tous les classements possibles, c'est-à-dire toutes les permutations de $[\![0\,..\,n-1]\!]$, on calcule pour chaque classement σ la valeur de Slater(T,σ) et on retient un classement qui donne la plus petite valeur de cette fonction.

Pour préparer l'écriture de cette fonction, on s'intéresse, pour un n fixé, à la liste des permutations de $[\![0\,..\,n-1]\!]$ triée par ordre lexicographique croissant. Par exemple, pour $n=3$, cette liste est $(0,1,2),(0,2,1),(1,0,2),(1,2,0),(2,0,1),(2,1,0)$.

On suppose fournie une fonction `next_permutation` qui recevra en paramètre, sous la forme d'un tableau d'entiers `sigma`, une permutation σ des entiers $[\![0\,..\,n-1]\!]$. Si cette permutation n'est pas la permutation $(n-1,n-2,\ldots,1,0)$, c'est-à-dire si ce n'est pas la dernière permutation dans le tri considéré, la fonction modifie le contenu du tableau `sigma` pour que `sigma` contienne, après l'appel à la fonction, la permutation suivante dans l'ordre lexicographique et retourne la valeur `true`; dans le cas contraire, elle renvoie la valeur `false` et ne transforme pas `sigma`.

```
val next_permutation : int array -> bool = <fun>
```

```
bool next_permutation(int sigma[], int n);
```

15. Écrire une fonction `order_Slater` telle que, si `T` est une matrice de booléens codant un tournoi T d'ordre n, alors `order_Slater t` (en OCaml) ou `order_Slater(t, n)` (en langage C) retourne un tableau de longueur n (un objet de type `int Array` en OCaml, un pointeur vers un tableau d'entier alloué par la fonction en langage C) contenant un classement de Slater pour T.

```
val order_Slater : bool array array -> int Array = <fun>
```

```
int* order_Slater(bool** t, int n);
```

2 Coloration optimale d'un graphe triangulé

2.1 Introduction et définitions

Une k-coloration propre d'un graphe non-orienté $G = (V, E)$ avec k couleurs est une application $c : V \mapsto [\![1 \,..\, k]\!]$ telle que pour toute arête $(v_i, v_j) \in E$, $c(v_i) \neq c(v_j)$. Une k-coloration est dite optimale s'il n'existe pas de $(k-1)$ coloration propre du graphe. On qualifique alors k de *nombre chromatique* du graphe.

Dans ce problème, on s'intéresse à un ensemble de graphes particuliers, dits *triangulés*, et à une méthode permettant d'obtenir simplement une k-coloration propre optimale.

On demande ici **de soigner les démonstrations**, en respectant les termes définis par le sujet, et en étant aussi clair, rigoureux et concis que possible dans les arguments.

Un *chemin de longueur k* dans un graphe $G = (V, E)$ est défini comme une suite de sommets $v_1, v_2, \ldots, v_k$ tel que pour tout $i \in [\![1 \,..\, k-1]\!]$, les sommets v_i et v_{i+1} sont voisins dans G. Un chemin est dit *élémentaire* si ses sommets sont tous distincts, à l'exception possible du premier et du dernier sommet du chemin.

Un chemin est dit *fermé* si le premier et le dernier sommet sont les mêmes. Un *cycle de longueur k* est un chemin de longueur $k \geqslant 3$ élémentaire et fermé.

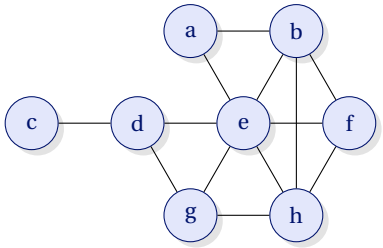
Un graphe est dit *triangulé* si dans tout cycle de longueur supérieure ou égale à 4, il existe deux sommets non adjacents dans ce cycle qui sont adjacents dans le graphe.

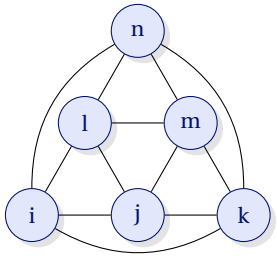
Soit $c = v_1, v_2, \ldots, v_k$ un chemin dans un graphe $G = (V, E)$. Une *corde* pour ce chemin est une arête $\{v_i, v_j\} \in E$ avec $|i - j| > 1$. Un chemin est dit sans corde s'il n'existe pas de telle arête pour ce chemin. Ainsi, un graphe est triangulé si et seulement si tous ses cycles de longueur supérieure ou égale à 4 ont au moins une corde.

1. Montrer que s'il existe un chemin $c = v_1, v_2, \ldots, v_k$ dans $G = (V, E)$, alors il existe un chemin c' de v_1 à v_k ne passant que par des sommets du chemin c et qui soit sans corde.

2.2 Premiers exemples

On considère les deux graphes ci-dessous :





2. Les graphes ci-dessus sont-ils triangulés ?

Un sous-ensemble $V' \subseteq V$ des sommets de G et une *clique* si et seulement si les sommets de V' sont tous voisins deux à deux. Un sommet v du graphe G est dit *simplicial* si son voisinage est une clique.

3. Quels sont les sommets simpliciaux de ces graphes ?

2.3 Identification de sommets simpliciaux

4. Montrer que avec soin que, si $G = (V, E)$ admet un sommet simplicial v et si $G' = (V', E')$, le sous-graphe de G induit par $V' = V \setminus v$, est triangulé, alors G est triangulé.

On souhaite à présent montrer que tout graphe triangulé admet au moins un sommet simplicial. **Dans la suite du sujet,** G **est un graphe triangulé**.

5. Montrer que si G est une clique, alors tous ses sommets sont simpliciaux.

6. Montrer que si, dans $G = (V, E)$, $x \in V$ est adjacent à tous les autres sommets, alors chaque sommet y simplicial dans le sous-graphe $G' = (V', E')$ induit par $V' = V \setminus x$ l'est aussi dans G.

7. Montrer, réciproquement, que si, dans $G = (V, E)$, $x \in V$ est adjacent à tous les autres sommets, alors chaque sommet y simplicial dans G l'est aussi dans le sous-graphe $G' = (V', E')$ induit par $V' = V \setminus x$.

On suppose à présent que $x \in V$ n'est pas adjacent à tous les sommets, et on considère

- T l'ensemble des sommets les plus éloignés de x,
- H une composante connexe de T,
- U l'ensemble des voisins de H dans G (les sommets $v \in V \setminus H$ tels qu'il existe un sommet de H voisin de v),
- et enfin Q la composante connexe de $G' = (V', E')$ induit par $V' = V - U$ contenant x.

8. Déterminer les ensembles T, H, U et Q pour le graphe d'ordre 8 pris en exemple à la question 1 de la section 2.3.

9. Pour un graphe G triangulé quelconque, montrer que U est une clique dans G.

10. En déduire l'existence d'un sommet simplicial dans G.

2.4 Identification de sommets simpliciaux

On souhaite à présent identifier l'un des sommets simpliciaux d'un graphe G = (V, E) triangulé. Pour ce faire, on considère l'algorithme suivant :

Algorithme 1 : Maximum Cardinal Search

Données : Graphe non-orienté G = (V, E)

pour tous $v \in V$ **faire**
    $\ell[v] \leftarrow 0$
    $\pi[v] \leftarrow$ Nil
fin
$F \leftarrow V$
tant que $F \neq \emptyset$ **faire**
    Soit $v \in F$ tel que $\ell[v]$ est maximal
    pour tous *voisin* v' *de* v *dans* F **faire**
        $\ell[v'] \leftarrow \ell[v'] + 1$
        $\pi[v'] \leftarrow v$
    fin
    $F \leftarrow F \setminus \{v\}$
fin

11. Reproduire les deux graphes en exemple, en indiquant sur chaque nœud un ordre de visite possible par l'algorithme MCS, et en mettant en évidence les arêtes de la forme $(i, \pi(i))$ pour le π obtenu à l'issue de cette exécution de l'algorithme.

Soit G = (V, E) un graphe triangulé, et $f : S \mapsto [\![1 \,..\, n]\!]$, une numérotation des sommets selon leur ordre de traitement par l'algorithme MCS. Soit $c = u, v_1, v_2, \ldots, v_k, w$ un chemin tel que $\forall i \in [\![1 \,..\, n]\!]$, $f(u) < f(v_i)$ et $f(w) < f(v_i)$.

12. Montrer que le chemin c a nécessairement une corde.

13. En déduire que le dernier sommet visité par l'algorithme MCS est simplicial.

14. Proposer un algorithme glouton de reconnaissance des graphes triangulés.

2.5 Coloration du graphe

15. Proposer un algorithme glouton de coloration propre optimale d'un graphe triangulé G.

16. Quelle relation a-t-on entre le nombre chromatique d'un graphe triangulé G et la taille maximale d'une clique de G ? On justifiera la réponse.