Devoir surveillé 3 b

Mines 2011 & 2016, graphes

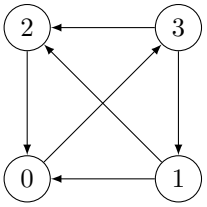
Option informatique MP1, MP2 & MP3

I Mines 2011 : ordres pour un tournoi

On s'intéresse à un jeu nommé J pour lequel les parties se jouent entre deux joueurs ; pour chaque partie du jeu J, il y a un gagnant et un perdant, il n'y a pas de match nul.
On considère une compétition C du jeu J effectuée par n joueurs. Les joueurs sont identifiés par des numéros allant de 0 à $n-1$. Chaque joueur joue une et une seule fois au jeu J contre chaque autre joueur. Le résultat de cet ensemble de $\frac{n(n-1)}{2}$ parties est représenté par une une matrice carrée `t` à coefficients booléens telle que `t.(i).(j)` vaut `true` si le joueur i a gagné contre le joueur j et `false` sinon. Par convention, `t.(i).(i)` vaut `false`
On appelle **tournoi** cette matrice, n est l'**ordre** du tournoi.
Un tournoi est la représentation par matrice d'adjacence d'un graphe à n sommet.

Par exemple le tournoi

```
let t4 = [|[|false; false; false;  true|];
           [| true; false;  true; false|];
           [| true; false; false; false|];
           [|false;  true;  true; false|]|];;
```



est l'implémentation du graphe G_4 ci-contre.
Dans ce tournoi :

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

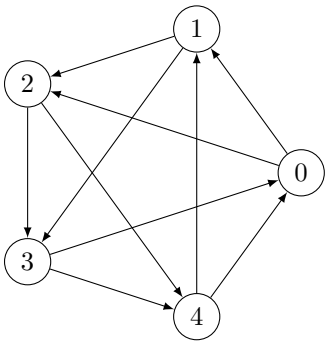


FIGURE 1 – Graphe G_5

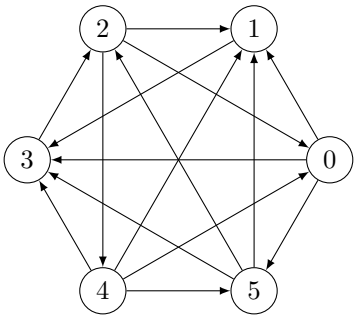


FIGURE 2 – Graphe G_6

Dans la suite, on utilisera les tournois `t5` et `t6` représentant les graphes ci-dessus.

On appelle **classement** d'ordre n toute permutation des entiers $0, 1, 2, \ldots, n-1$.
Un classement σ d'ordre n sera noté $\big(\sigma(0), \sigma(1), \ldots, \sigma(n-1)\big)$.
Il sera codé par un tableau (`array`) d'entiers de dimension n.

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 apparait 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.
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.

I.1 Méthode de Copeland

Dans un tournoi `t` d'ordre n, le **score** de i (pour $0 \leqslant i < n$) est le degré sortant de i, c'est le nombre de parties gagnées par le joueur i. Par exemple, si la compétition est décrite par `t4`, les scores des joueurs 0, 1, 2 et 3 sont respectivement égaux à 1, 2, 1 et 2.
Un classement est appelé **classement de Copeland** pour le tournoi `t` si les scores des joueurs dans ce classement sont décroissants au sens large.
Par exemple, le classement $(1, 3, 0, 2)$ est un classement de Copeland pour le tournoi `t4`.

Exercice 1 -
Indiquer un classement de Copeland pour le tournoi `t6`.

Exercice 2
Écrire une fonction `calculer_scores : bool array array -> int array` qui renvoie, pour un tournoi d'ordre n, le tableau des scores des n joueurs.
Indiquer la complexité de cette fonction.

Exercice 3
Écrire une fonction `classement_Copeland : bool array array -> int array` qui renvoie un tableau d'entiers contenant un classement de Copeland pour le tournoi passé en paramètre.
Indiquer la complexité de cette fonction.

I.2 Valeur de Slater d'un classement

Soient `t` 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 est 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`, notée Slater$(\mathtt{t}, \sigma)$ le nombre de parties de C contredites par σ. Autrement dit, Slater$(\mathtt{t}, \sigma)$ est le nombre de couples (i, j) vérifiant simultanément

$$0 \leqslant i < j < n \text{ et } \mathtt{t}.(\mathtt{p}).(\mathtt{q}) = \mathtt{false} \text{ pour } p = \sigma(i),\ q = \sigma(j)$$

À titre d'exemple, le classement $\sigma_4 = (1, 3, 2, 0)$ a une valeur de Slater de 2 pour le tournoi `t4` : les parties contredites par le classement étant la partie entre 1 et 3 et la partie entre 0 et 3.

Exercice 4
Calculer Slater($\texttt{t5}, \sigma_5$) pour $\sigma_5 = (2, 4, 0, 1, 3)$.

Exercice 5
Calculer Slater($\texttt{t6}, \sigma_6$) si σ_6 est le classement de Copeland pour `t6` déterminé à la question 1

Exercice 6
Écrire une fonction `valeur_Slater : bool array array -> int array -> int` qui renvoie Slater($\texttt{t}, \sigma$) pour un tournoi `t` et un tableau codant un classement σ passés en paramètres.
Indiquer la complexité de cette fonction.

I.3 Indice de Slater d'un tournoi

On appelle **indice de Slater** d'un tournoi `t` d'ordre n le minimum de Slater Slater($\texttt{t}, \sigma$) sur l'ensemble des classements σ d'ordre n. On le note $s(\texttt{t})$.
Un **classement de Slater** pour un tournoi `t` est un classement σ d'ordre n vérifiant Slater($\texttt{t}, \sigma$) = $s(\texttt{t})$. Ainsi, un classement de Slater pour `t` contredit un minimum de parties de la compétition représentée par le tournoi `t`.

Exercice 7
Calculer $s(\texttt{t4})$ et indiquer un classement de Slater pour `t4`.

Si `t` est un tournoi représentant un graphe G, un **circuit** pour `t` est un circuit de G c'est-à-dire une suite finie $(i_1, i_2, \ldots, i_p)$ avec $p \geqslant 3$ tel que (i_k, i_{k+1}) est une arête de G[1] pour tout $k \in \{1, 2, \ldots, p-1\}$ ainsi que (i_p, i_0).
Si p vaut 3, on dit qu'il s'agit d'un triangle de `t`. On dit que deux circuits de `t` sont arc-disjoints s'ils n'ont pas d'arc en commun.
Par exemple, dans `t5`, les deux circuits $(0, 1, 3)$ et $(0, 2, 3, 4)$ sont arc-disjoints.

Exercice 8
On suppose qu'il existe m circuits 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.

Exercice 9
On considère un tournoi `t` d'ordre n, un classement σ d'ordre n et un ensemble de m circuits deux à deux arc-disjoints dans le tournoi `t`. On suppose que l'on a Slater($\texttt{t}, \sigma$) = m.
Indiquer alors ce qu'on peut dire de l'indice de Slater de `t` et du classement σ.

Exercice 10
Calculer $s((t6)$ et indiquer un classement de Slater pour `t6`. Déduire de ce résultat qu'un classement de Slater pour un tournoi `t` n'est pas toujours un classement de Copeland.

1. On rappelle que cela signifie que i_k a battu i_{k+1}.

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`.

Exercice 11
Indiquer un ensemble maximal de triangles deux à deux arc-disjoints de `t5`.

On veut calculer le cardinal d'un ensemble maximal de triangles deux à deux arc-disjoints d'un tournoi `t` d'ordre n. On admet que, pour cela, il suffit de construire 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.

Exercice 12
Écrire une fonction `compter_triangles : bool array array -> int` qui renvoie 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, on ne justifiera pas cette complexité.

I.4 Méthode de Slater

Pour déterminer un classement de Slater pour `t`,

- on énumère tous les classements possibles, c'est-à-dire les permutations de $\{0, 1, \dots, n-1\}$,
- on calcule pour chaque classement σ la valeur de Slater$(\texttt{t}, \sigma)$
- et on retient un classement qui donne la plus petite valeur de cette fonction.

Pour énumérer les permutations, on peut utiliser l'algorithme de Johnson qui détermine la permutation qui suit une permutation σ dans l'ordre lexicographique.

- On détermine le plus grand entier i tel que $\sigma(i) > \sigma(i-1)$; on admet qu'un tel entier existe si σ n'est pas la dernière permutation, $(n-1, n-2, \dots, 1, 0)$.
- On calcule l'entier $j \geqslant i$ tel que $\sigma(j) > \sigma(i-1) > \sigma(j+1)$ ou $j = n-1$ si $\sigma(n-1) > \sigma(i-1)$
- On échange le valeurs de $\sigma(i-1)$ et de $\sigma(j)$.
- On retourne la portion de tableau entre i et $n-1$, ce qui la rend croissante.

Exercice 13
Indiquer les huit premières permutations qui suivent la permutation $(1, 2, 5, 4, 0, 3)$.
On ne justifiera pas la réponse.

Exercice 14
Écrire une fonction `permutation_suivante : int array -> int array` qui effectue les opérations décrites ci-dessus.

Exercice 15
Écrire une fonction `classement_Slater : bool array array` qui renvoie un vecteur de longueur n contenant un classement de Slater pour `t`.

II Mines 2016 : graphe du web

Le World Wide Web, ou Web, est un ensemble de pages Web (identifiées de manière unique par leurs adresses Web, ou URL pour Uniform Resource Locators, de la forme http ://mines-ponts.fr/index.php) reliées les unes aux autres par des hyperliens.
Le Web est souvent modélisé comme un graphe orienté dont les sommets sont les pages Web et les arcs les hyperliens entre pages.
Le Web étant potentiellement infini, on s'intéresse à des sous-graphes du Web obtenus en naviguant sur le Web, c'est-à-dire en le parcourant page par page, en suivant les hyperliens d'une manière bien déterminée. Ce parcours du Web pour en collecter des sous-graphes est réalisé de manière automatique par des logiciels autonomes appelés Web crawlers ou crawlers en anglais, ou collecteurs en français.

II.1 Fonctions utilitaires

Exercice 16
Coder une fonction `aplatir : ('a * 'a list) list -> 'a list`, telle que, si `liste` est une liste de couples $[(x_1, l_{x_1}); \ldots; (x_n, l_{x_n})]$, où chaque x_i est un élément de type `'a`, et l_{x_i} une liste d'éléments de type `'a` de la forme $[y_{i,1}; \ldots; y_{i,k_i}]$, `aplatir liste` est une liste d'éléments de type `'a` : $[x_1; y_{1,1}; \ldots; y_{1,k_1}; x_2; y_{2,1}; \ldots; y_{2,k_2}; \ldots; x_n; y_{n,1}; \ldots; y_{n,k_n}]$.

Exercice 17
Coder une fonction `tri_fusion : (a * b) list -> (a * b) list` triant une liste de couples (x, y) par ordre décroissant de la valeur de la seconde composante y de chaque couple en utilisant l'algorithme de tri fusion. Quelle est la complexité de cet algorithme ?

On va utiliser dans la suite de l'exercice un type de données **persistant** `dictionnaire` qui permet de stocker des couples formés d'une chaîne de caractères (une clef) et d'un entier (une valeur). On dit que le dictionnaire associe la valeur à la clef. A chaque clef présente dans le dictionnaire est associée une seule valeur.
Les fonctions suivantes sont supposées être prédéfinies : n est le nombre d'entrées du dictionnaire.

`dictionnaire_vide : unit -> dictionnaire` qui crée un nouveau dictionnaire vide.

`ajoute : string -> int -> dictionnaire -> dictionnaire`
qui renvoie un nouveau dictionnaire identique au dictionnaire initial, sauf qu'un couple `(clef,valeur)` y a été ajouté. Cette fonction s'exécute en temps $O\big(\log(n)\big)$.

`contient : string -> dictionnaire -> bool`
qui renvoie un booléen indiquant s'il y a un couple dont la clef est `clef` dans le dictionnaire `dict`. Cette fonction s'exécute en temps $O\big(\log(n)\big)$.

`valeur : string -> dictionnaire -> int`
qui renvoie la valeur associée à la clef `clef` dans le dictionnaire `dict`. Cette fonction s'exécute en temps $O(\log n)$ et ne peut être appelée que si la clef `clef` est présente dans le dictionnaire.

On suppose pour la suite de l'exercice que le type de données `dictionnaire` est prédéfini ; on ne demande pas de l'implémenter.

Exercice 18
Coder `unique : string list -> string list * dictionnaire`, qui est telle que `unique liste` renvoie un couple `(liste', dict)` où `liste'` est la liste des chaînes de caractères distinctes de `liste` (dans l'ordre de leur première occurrence) et où `dict` associe à chaque chaîne de caractères sa position dans `liste'` (en numérotant à partir de 0).

Par exemple, l'appel `unique ["x"; "zz"; "x"; "x"; "zz"; "yt"]` renvoie le couple formé de la liste `["x"; "zz"; "yt"]` et d'un dictionnaire associant à `"x"` la valeur `0`, à `"zz"`, `1` et à `"yt"`, `2`.

Exercice 19
Quelle est la complexité de la fonction `unique` en terme de la longueur n de la liste `liste` en argument et du nombre m d'éléments distincts dans la liste `liste` ? Justifier la réponse.

II.2 Crawler simple

Nous allons maintenant implémenter un crawler simple en OCaml. On suppose fournie une fonction `recupere_liens : string -> string list`prenant en argument l'URL d'une page Web p et renvoyant la liste des URL des pages q pour lesquelles il existe un hyperlien de p à q, dans l'ordre lexicographique.

Pour illustrer le comportement de cette fonction, nous considérons un exemple de mini-graphe du Web à six pages et neufs hyperliens. On remarque qu'il y a deux liens de "p4" vers "p5".

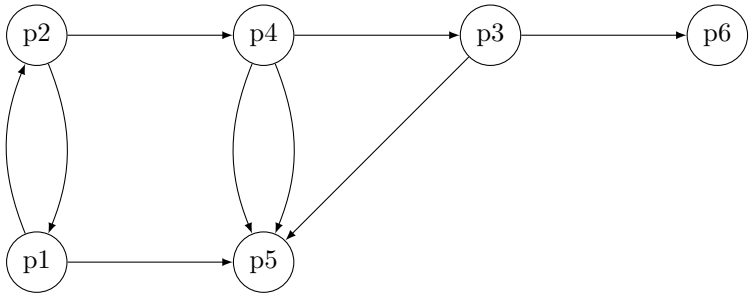


FIGURE 3 – Un exemple de Web, W_0

Dans cette représentation, p_1, p_2, etc., sont les URL de pages Web (simplifiées pour l'exemple), et les arcs représentent les hyperliens entre pages Web.
Un appel à `recupere_liens "p1"` retourne la liste `["p2"; "p5"]`.
Un appel à `recupere_liens "p4"` retourne la liste `["p3"; "p5"; "p5"]`.

Un **crawler** est un programme qui, à partir d'un entier n et d'une URL, parcourt le graphe du Web en visitant progressivement les pages dont les liens sont présents dans chaque page rencontrée. A chaque nouvelle page, si celle-ci na pas déjà été visitée, tous ses hyperliens sont récupérés et ajoutés à une liste de liens à traiter. Le processus s'arrête quand n pages ont été visitées. Le résultat renvoyé par le crawler, un **crawl**, est une liste de longueur au plus n de couples (v, l) où v est l'URL d'une page visitée (les pages apparaissant dans l'ordre où elles ont été visitées) et l la liste des liens récupérés sur la page v.
On utilisera une variable de type dictionnaire pour se souvenir des pages déjà visitées.

Exercice 20
Écrire une fonction `crawler_bfs : int -> string -> (string * string list)list` qui code un crawler qui parcourt le graphe du Web en suivant un parcours en largeur d'abord (breadth-first search), c'est-à-dire en visitant en priorité les pages rencontrées le plus tôt dans l'exploration.

Par exemple, sur W_0, `crawler_bfs 4 "p1"` pourra renvoyer le résultat :

```
["p1", ["p2"; "p5"];
 "p2", ["p1"; "p4"];
 "p5", [];
 "p4", ["p3"; "p5"; "p5"]]
```

Exercice 21
Écrire une fonction `crawler_dfs : int -> string -> (string * string list)list` qui code un crawler qui parcourt le graphe du Web en suivant un parcours en longueur d'abord (depth-first search), c'est-à-dire en visitant en priorité les pages rencontrées le plus récemment dans l'exploration.

Par exemple, sur W_0, `crawler_dfs 4 "p1"` pourra renvoyer le résultat :

```
["p1", ["p2"; "p5"];
 "p2", ["p1"; "p4"];
 "p4", ["p3"; "p5"; "p5"];
 "p3", ["p5"; "p6"]]
```

Exercice 22
Coder une fonction `construit_graphe : (string * string list)list -> string list * int vect vect` telle que si *crawl* est le résultat renvoyé par un crawler alors `construit_graphe crawl` renvoie un couple (l, G) où l est une liste de toutes les URL de pages contenues dans la liste *crawl* et G est la matrice d'adjacence du sous-graphe partiel du Web restreint aux pages de la liste l : $G_{i,j}$ est le nombre de liens découverts dans le crawl de la page d'indice i dans l vers la page d'indice j dans l.
On fera commencer les indices à 0.

Par exemple, sur W_0, si `crawl` est le résultat de `crawler_bfs 4 "p1"`, alors `construit_graphe crawl` doit renvoyer :

```
["p1"; "p2"; "p5"; "p4"; "p3"], [|[|0; 1; 1; 0; 0|];
                                  [|1; 0; 0; 1; 0|];
                                  [|0; 0; 0; 0; 0|];
                                  [|0; 0; 2; 0; 1|];
                                  [|0; 0; 0; 0; 0|]|]
```

En particulier :

- p_3 apparaît même s'il na pas été visité dans le crawl ;
- p_6 n'apparaît pas car il na pas été découvert dans le crawl ;
- l'hyperlien de p_3 à p_5 n'apparaît pas car p_3 na pas été visité.

II.3 Calcul de PageRank

PageRank est une manière d'affecter un score à l'ensemble des pages du Web, imaginée par Sergey Brin et Larry Page, les fondateurs du moteur de recherche Google. L'introduction de PageRank a révolutionné la technologie des moteurs de recherche sur le Web. Nous allons maintenant implémenter le calcul de PageRank.
Étant donnée une partie du Web (où l'ensemble des pages est indexé entre 0 et $n-1$), la matrice de surf aléatoire dans cette partie du Web est la matrice M de taille $n \times n$ définie comme suit :

- s'il n'y a aucun lien depuis une page Web d'indice i, alors pour tout j, $M_{ij} := \frac{1}{n}$.
- Sinon, s'il y a k_i liens depuis la page Web d'indice i, alors pour tout j, on a $M_{i,j} := (1-d) \times \frac{G_{i,j}}{k_i} + \frac{d}{n}$, où $G_{i,j}$ est le nombre de liens depuis la page d'indice i vers la page d'indice j et d est un nombre réel fixé appartenant à $[0, 1]$ (on prend souvent $d = 0,15$).

Cette matrice peut être vue comme décrivant la marche aléatoire d'un surfeur sur le Web. à chaque fois que celui-ci visite une page Web :

- Si cette page ne comporte aucun lien, il visite une page Web arbitraire, choisie aléatoirement de façon uniforme.

- Si cette page comporte au moins un lien, il visite avec une probabilité égale à $1-d$ un des liens sortants de cette page, et avec une probabilité égale à d une page Web arbitraire, choisie aléatoirement de façon uniforme.

Exercice 23
Coder `surf_aleatoire : float -> int vect vect -> float vect vect` telle que si d est un nombre entre 0 et 1, et si G est la matrice d'adjacence d'un sous-graphe partiel du Web, alors `surf_aleatoire d G` renvoie la matrice M de surf aléatoire dans ce sous-graphe.

Exercice 24
Coder `multiplie : float vect -> float vect vect -> float vect`, une fonction prenant en argument un vecteur ligne v de taille n et une matrice M de taille $n \times n$ et renvoyant le vecteur ligne w de taille n résultant du produit de v par la matrice M : $w = vM$.

En d'autres termes, pour tout j, $w_j = \sum_{i=0}^{n-1} v_i M_{ij}$.

Le PageRank des pages d'un sous-graphe du Web à n pages est la limite, quand p tend vers $+\infty$, de $v_0.M^p$ produit du vecteur ligne v_0 dont toutes les composantes valent $\frac{1}{n}$ par les puissances de la matrice de surf aléatoire M de ce sous-graphe.

En pratique, on se fixe un réel h strictement positif (par exemple, $h = 10^{-4}$)
et on calcule la suite des $v^{(p)}v_0.M^p$ jusqu'à ce que [2] $\|v^{(p+1)} - v^{(p)}\|_1 \leqslant h$.
On renvoie alors le vecteur $v^{(p+1)}$, considéré comme le vecteur des scores de PageRank.
On peut montrer [3] que l'algorithme termine dès lors que d est strictement positif.

PageRank est utilisé pour affecter un score d'importance aux pages du Web. Le vecteur de scores v retourné par l'algorithme de PageRank donne dans v_i le score d'importance de la page d'indice i. Les pages de plus haut score de PageRank sont considérées comme les plus importantes.

Exercice 25
Coder `pagerank : float -> float vect vect -> float vect`, une fonction prenant en argument un nombre $h > 0$ et une matrice M de surf aléatoire d'un sous-graphe du Web et renvoyant le vecteur des scores de PageRank pour h et M.

Exercice 26
Coder `calcule_pagerank : float -> float -> (string * string list)list -> (string * float)list` telle que `calcule_pagerank d h crawl` renvoie une liste de couples (u, s), un couple pour chaque URL découverte dans le crawl crawl, triée par valeur décroissante de s, où u est l'URL de cette page et s son score de PageRank.
Ici, d et h sont les deux paramètres nécessaires au calcul de la matrice de surf aléatoire et du PageRank respectivement.

Dans les deux dernières questions appel aux fonctions `multiplie` et `tri_fusion` et à l'ensemble des fonctions développées dans les questions précédentes.

2. On utilise ici la norme $\|.\|_1$ pour simplifier les calculs.
3. Avec l'aide du théorème de Perron-Frobenius

Solutions

Solution de l'exercice 1 -
Les scores des joueurs de 0 à 5 sont 3, 1, 3, 1, 4 et 3.
Un classement possible de `t6` est donc $\sigma_6 = (4, 5, 2, 0, 1, 3)$.

Solution de l'exercice 2

```
let calculer_scores t =
   let n = Array.length t in
   let s = Array.make n 0 in
   for i = 0 to (n-1) do
      for j = 0 to (n-1) do
         if t.(i).(j)
         then s.(i) <- s.(i) + 1 done done;
    s;;
```

La complexité est un $\mathcal{O}(n^2)$.

Solution de l'exercice 3
On trie le tableau obtenu. Comme la complexité est déjà quadratique, on peut utiliser un tri de complexité quadratique, par exemple un tri par insertion.

```
let classement_Copeland t =
  let s = calculer_scores t in
  let n = Array.length s in
  let c = Array.init n (fun i -> i) in
  for i = 1 to (n-1) do
    let j = ref i in
    while !j > 0 && s.(c.(!j -1)) < s.(i) do
      c.(!j) <- c.(!j-1);
      j := !j - 1 done;
    c.(!j) <- i done;
  c;;
```

La complexité reste un $\mathcal{O}(n^2)$.

Solution de l'exercice 4
Slater($\texttt{t5}, \sigma_5$) vaut 4, les parties contredites sont $(2, 0)$, $(2, 1)$, $(4, 3)$ et $(0, 3)$.

Solution de l'exercice 5
Slater($\texttt{t5}, \sigma_5$) vaut 3, les parties contredites sont $(4, 2)$, $(5, 0)$ et $(2, 3)$.

Solution de l'exercice 6

```
let valeur_Slater t c =
   let n = Array.length c in
   let v = ref 0 in
   for i = 0 to n-2 do
      for j = (i+1) to (n-1) do
         if not t.(c.(i)).(c.(j))
         then incr v done done;
   !v;;
```

Solution de l'exercice 7
Dans `t4` aucun joueur n'est battu par tout le monde donc aucun classement ne peut avoir de dernier terme sans qu'il y ait de contradiction : $s(\texttt{t4}) \geqslant 1$.
$\sigma'_4 = (3,1,2,0)$ vérifie $\text{Slater}(\texttt{t}_4, \sigma_4) = 1$ donc $s(\texttt{t4}) = 1$ et σ'_4 est un classement de Slater pour `t4`.

Solution de l'exercice 8
σ est un classement de `t`.
À chaque circuit $(i_1, i_2, \ldots, i_p)$ de `t` on associe le premier indice i_k apparaissant dans σ. Pour $k \neq 1$, (i_{k-1}, i_k) est une arête avec i_{k-1} classé après i_k, c'est une contradiction. Pour $k = 1$, (i_p, i_1) est une contradiction. Ainsi chaque circuit contient une contradiction parmi ses arâtes.
m circuits deux à deux arc-disjoints impliquent donc m contradictions : $s(\texttt{t}) \geqslant m$.

Solution de l'exercice 9
Dans ce cas m est un minorant de $s(\texttt{t})$ d'après l'exercice précédent et un majorant de $s(\texttt{t})$ d'après la définition. On a donc $s(\texttt{t}) = m$ et σ est classement de Slater pour `t`.

Solution de l'exercice 10
On a trouvé un classement de Copeland avec 3 contradictions donc $s(\texttt{t6}) \leqslant 3$
On peut trouver 2 circuits arcs-disjoints : $(2,4,3)$ et $(0,5,2)$ donc $s(\texttt{t6}) \geqslant 2$.
Si on veut trouver un classement avec 2 contradictions, celles-ci sont des arêtes des circuits donc 1 ne peut pas donner de contradiction : on doit le placer juste avant 3. Ainsi la contradiction du circuit $(2,4,3)$ sera $(3,2)$: le classement doit contenir $(2,4,1,3)$ dans l'ordre. De même 4 ne peut pas induire de nouvelle contradiction donc doit être avant 5 et 0. On aboutit au classement $(2,4,0,5,3,1)$ de valeur de Slater 2 donc minimal.
Ce classement n'est pas un classement de Copeland car 4, de score maximal 4 est précédé de 2 de score 3.

Solution de l'exercice 11
Les triangles de `t5` sont $(0,1,3)$, $(0,2,3)$, $(0,2,4)$, $(1,2,4)$ et $(1,3,4)$.
Seules les paires $\big((0,1,3),(0,2,4)\big)$, $\big((0,1,3),(1,2,4)\big)$, $\big((0,2,3),(1,2,4)\big)$, $\big((0,2,3),(1,3,4)\big)$ et $\big((0,2,4),(1,3,4)\big)$ sont arcs-disjointes et on constate qu'on ne peut pas les augmenter. Chacune est un exemple d'ensemble maximal de triangles deux à deux arc-disjoints.

Solution de l'exercice 12
On recopiera la matrice `t`

```
let copie_matrice m =
   let n = Array.length m in
   let p = Array.length m.(0) in
   let mm = Array.make_matrix n p m.(0).(0) in
   for i = 0 to (n-1) do
      for j = 0 to (p-1) do
         mm.(i).(j) <- m.(i).(j) done done;
   mm;;
```

```
let compter_triangles t =
   let n = Array.length t in
   let tt = copie_matrice t in
   let tri = ref 0 in
   for i = 0 to (n-1) do
      for j = 0 to (n-1) do
         for k = 0 to (n-1) do
            if tt.(i).(j) && tt.(j).(k) && tt.(k).(i)
            then begin incr tri;
                 tt.(i).(j) <- false;
                 tt.(j).(k) <- false;
                 tt.(k).(i) <- false end done done done;
   !tri;;
```

Solution de l'exercice 13
L'algorithme donne successivement
$(1, 2, 5, 4, 3, 0)$
$(1, 3, 0, 2, 4, 5)$
$(1, 3, 0, 2, 5, 4)$
$(1, 3, 0, 4, 2, 5)$
$(1, 3, 0, 4, 5, 2)$
$(1, 3, 0, 5, 2, 4)$
$(1, 3, 0, 5, 4, 2)$
$(1, 3, 2, 0, 4, 5)$

Solution de l'exercice 14
On commence par l'échange classique dans un tableau.

```
let echange a b t =
  let temp = t.(a) in
  t.(a) <- t.(b);
  t.(b) <- temp;;
```

```
let permutation_suivante s =
  let n = Array.length s in
  let i = ref (n-1) in
  while !i > 0 && s.(!i) < s.(!i-1) do decr i done;
  if !i = 0
  then false
  else begin let j = ref !i in
    while !j < n - 1 && s.(!j+1) > s.(!i-1) do incr j done;
    echange  (!i - 1) !j s;
    for k = !i to (n + !i - 1)/2 do
      echange  k (n - 1 - (k - !i)) s done;
    true end;;
```

Solution de l'exercice 15

```
let classement_Slater t =
  let n = Array.length t in
  let s = Array.init n (function i -> i) in
  let mini = ref (valeur_Slater t s) in
  let s_mini = ref (Array.copy s) in
  while  permutation_suivante s do
    let v = valeur_Slater t s in
    if v < !mini
    then begin mini := v; s_mini := Array.copy s end done;
  !s_mini;;
```

Solution de l'exercice 16

```
let rec aplatir liste =
   match liste with
  |[] -> []
  |(x, l)::q -> (x::l)@(aplatir q);;
```

Solution de l'exercice 17

On commence par découper une liste en 2 listes de tailles presqu'égales

```
let rec decouper l =
   match l with
  |[] -> [], []
  |[x] -> [x], []
  |x::y::q -> let l1, l2 = decouper q in
              x::l1, y::l2;;
```

On calcule la fusion de deux listes triées.

```
let rec fusion l1 l2 =
   match (l1,l2) with
  |[],_ -> l2
  |_,[] -> l1
  |(a, b)::q1,(c, d)::q2 when b >= d -> (a, b)::(fusion q1 l2)
  |(a, b)::q1,(c, d)::q2 -> (c,d )::(fusion l1 q2);;
```

Lafonction de tri a une complexité en $\mathcal{O}\big(n\log_2(n)\big)$.

```
let rec tri_fusion l =
   match l with
  |[] -> []
  |[x] -> [x]
  |_  -> let l1, l2 = decouper l in
         fusion (tri_fusion l1) (tri_fusion l2);;
```

Solution de l'exercice 18
On accumule les nouveaux éléments dans une liste, leur ordre dans la liste sera inversé d'où l'utilisation de `List.rev` à la fin pour donner les indices corrects.

```
let unique l =
    let rec aux a_faire fait dict k =
      match a_faire with
      |[] -> List.rev fait, dict
      |c::reste when contient c   dict -> aux reste fait dict k
      |c::reste -> aux reste (c::fait) (ajoute c k dict) (k+1) in
   aux l [] (dictionnaire_vide ()) 0;;
```

Solution de l'exercice 19
Pour chaque élément de la liste initiale on teste son appartenance au dictionnaire et on peut l'ajouter au dictionnaire. Chacune de ces opération a un coût en $\mathcal{O}(m)$ car le dictionnaire contient au plus m éléments. Comme le retournement de la liste est de complexité linéaire la complexité est en $\mathcal{O}\big(n\log(m)+m\big) = \mathcal{O}\big(n\log(m)\big)$ car on a $m \leqslant m$.

Solution de l'exercice 20
L'énoncé parle de "*liste de liens à traiter*" ; on peut donc implémenter la file d'attente par une liste à laquelle on concatène les voisins. On pouvait, bien sur utiliser une file d'attente que l'on définirait. Ici je choisis d'utiliser deux listes : la première contient les sommet que l'on traite, la seconde contient les sommets à traiter ensuite.
Comme la structure qui permet de retenir les sommets déjà visités est un dictionnaire persistant, on doit la jouter aux paramètres de la fonction auxiliaire. Celle-ci a donc pour paramètres :

- le nombre de sommets restant à explorer, `k`
- la liste des sommets à traiter, `now`
- la liste des sommets au rang suivant, **`next`**
- le dictionnaire des sommets vus, `vus`
- le crawl déjà construit, `cwl`

On bascule à **`next`** à la place de `now` quand `now` est vide, auquel cas on s'arrête.

```
let crawler_bfs n p0 =
   let rec aux k now next vus cwl =
      if k = 0 || (now = [] && next = []
      then cwl
      else begin match now with
           |[] -> aux k next [] vus cwl
           |c::reste when contient c vus -> aux k reste next vus
              cwl
           |c::reste -> let plus = recupere_liens c in
                        let vus' = ajoute c 0 vus in
                        aux (k-1) reste (plus@next) vus' ((c, plus
                           ) :: cwl)
           end
   in aux n [p0] [] (dictionnaire_vide ()) [];;
```

L'usage de structures de données persistantes rend difficile l'emploi de `List.iter`.

Solution de l'exercice 21
On remplace la file par une pile, modélisée par une liste unique.

```
let crawler_dfs n p0 =
   let rec aux k now vus cwl =
      match k, now with
      |0, _ -> cwl
      |_, [] -> cwl
      |_, c::reste when contient c vus -> aux k reste vus cwl
      |_, c::reste -> let plus = recupere_liens c in
                      let vus' = ajoute c 0 vus in
                      aux (k-1) (plus@reste) vus' ((c, plus) :: cwl)
   in aux n [p0] (dictionnaire_vide ()) [];;
```

Solution de l'exercice 22
On itère sur le crawl une itération sur les voisins.

```
let construit_graphe crawl =
   let som, dico = unique (aplatir crawl) in
   let n = List.length som in
   let g = Array.make_matrix n n 0 in
   let traiter c1 c2 =
      let i = valeur c1 dico in
      let j = valeur c2 dico in
      g.(i).(j) <- g.(i).(j) + 1 in
   List.iter (fun (c, l) -> List.iter (traiter c) l) crawl;
   som, g;;
```

Solution de l'exercice 23

```
let surf_aleatoire d g =
   let n = Array.length g in
   let m = Array.make_matrix n n 0.0 in
   for i = 0 to n-1 do
      let k = ref 0 in
      for j = 0 to n-1 do k:= !k + g.(i).(j) done;
         if !k = 0
         then for j = 0 to n-1 do
                 m.(i).(j) <- 1./.(float_of_int n) done
         else for j = 0 to n-1 do
                 m.(i).(j) <- (1. -. d) *. (float_of_int g.(i).(j)
                    ) /. (float_of_int !k)
                              +. d /. (float_of_int n) done;
   done;
   m;;
```

Solution de l'exercice 24

```
let multiplie v m =
    let n = Array.length v in
    let w = Array.make n 0. in
    for j=0 to n-1 do
       for i=0 to n-1 do
           w.(j) <- w.(j) +. v.(i) *. m.(i).(j) done done;
    w;;
```

Solution de l'exercice 25
On commence par la norme

```
let norme v w =
   let n = Array.length v in
   let k = ref 0. in
   for i = 0 to (n - 1) do k := !k +. abs_float (w.(i) -. v.(i))
      done;
   !k;;
```

```
let pagerank theta m =
   let n = Array.length m in
   let u0 = Array.make n (1.0 /. (float_of_int n)) in
   let rec aux v =
      let w = multiplie v m in
      if norme v w <= theta then w else aux w
   in aux u0;;
```

Solution de l'exercice 26

```
let calcule_pagerank d theta crawl =
   let s, g = construit_graphe crawl in
   let m = surf_aleatoire d g in
   let v = pagerank theta m in
   let rec construire liste i =
      match liste with
      |[] -> []
      |c::q -> (c, v.(i)) :: (construire q (i+1)) in
   tri_fusion (construire s 0);;
```