L'algorithme de test de planarité de R. E. Tarjan

Robert Cori [1]

La planarité ?

Savoir si un graphe peut être dessiné sans croisement dans le plan est un problème très simple à exposer : il s'agit, étant donnés un ensemble $X = \{x_1, x_2, \ldots, x_n\}$ de n *sommets* et un ensemble $E = \{e_1, e_2, \ldots, e_m\}$ de m *arêtes*, qui sont des paires d'éléments de X, de vérifier si on peut représenter les sommets par des points sur un plan et chaque arête $e_k = \{x_i, x_j\}$ par une courbe joignant x_i à x_j, de façon telle que les courbes ne se coupent pas. Depuis Euler on sait que ceci est impossible si $m > 3n-6$ et depuis Kuratowski qu'une telle chose est possible si et seulement si le graphe ne contient pas par réduction (en un sens un peu compliqué que l'on ne développera pas ici) l'un des deux graphes suivants :

— le graphe K_5 contenant toutes les 10 arêtes formées avec $\{x_1, x_2, x_3, x_4, x_5\}$,

— le graphe $K_{3,3}$ contenant toutes les 9 arêtes $\{x_i, x_j\}$ formées avec $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ telles que i est pair et j impair.

Chercher un algorithme efficace pour résoudre ce problème était une question ouverte à la fin des années 60 — il s'agissait alors de réaliser des circuits électriques ou des composants électroniques de grande taille (chaque couche d'un circuit imprimé doit correspondre à un ensemble de connexions électriques réalisées sans croisements).

1. Professeur émérite à l'université de Bordeaux, membre du LaBRI, UMR 5800 CNRS. http://www.labri.fr/perso/cori/

L'application du théorème de Kuratowski donnant un algorithme nécessitant à priori un nombre d'opérations en n^7, plusieurs chercheurs, comme Auslander et Parter [2], Lempel, Even et Cederbaum [8] et Demoucron, Malgrange et Pertuiset [3], se sont attelés à proposer des algorithmes. La complexité de ces algorithmes n'était pas évaluée rigoureusement, ce n'était pas à l'ordre du jour alors. L'efficacité se mesurait en fonction du temps mis par l'ordinateur pour résoudre le problème sur des graphes ayant de l'ordre de 10, 100 ou 1000 sommets.

À cette époque, à Paris (Institut de Programmation), il était beaucoup question dans ce cadre d'un *Contrat Graphes* sous la responsabilité de Jean Berstel, Claude Girault et Jean-François Perrot, pour lequel Alain Jacques et Claude Lenormand avaient implanté ces algorithmes en proposant des structures de données élaborées.

Quelques années plus tard la thèse de R. E. Tarjan, soutenue à Stanford en 1971, a révolutionné le domaine des algorithmes pour les graphes car elle proposait de nouvelles idées et une présentation précise des algorithmes qui permettait l'évaluation de leur complexité. Le résultat principal de cette thèse était un algorithme en temps dépendant linéairement du nombre de sommets qui testait si un graphe est planaire. Je me propose de présenter les grandes lignes de cet algorithme en insistant sur deux points que je considère comme les plus importants permettant d'assurer cette complexité : l'algorithme de parcours en profondeur et les « piles de bi-piles ».

Une version très simplifiée du problème

On suppose dans un premier temps que les sommets $x_1, x_2, \ldots x_n$ sont des points placés sur un cercle, dans cet ordre, que n est pair ($n = 2m$), qu'il y a m arêtes en plus de celles du cercle — des cordes donc —, et enfin que chacun des x_i apparaît dans une et une seule de ces paires. On pose la question de savoir si on peut relier les points par des cordes à l'intérieur du cercle sans que celles-ci se coupent. En terme de graphes on suppose que le graphe est cubique (chaque sommet a trois voisins) et que l'on en connaît un cycle Hamiltonien (un cycle passant une seule fois par tous les sommets) ; on demande alors s'il est représentable par un graphe planaire pour lequel le cycle Hamiltonien détermine une face de longueur n.

On pourrait tester que les cordes ne se coupent pas en les traitant deux à deux, ce qui donnerait un algorithme en $O(n^2)$. On propose ici un algorithme en $O(n)$ qui fait intervenir une pile d'entiers. Cet algorithme sera modifié dans la section suivante pour traiter un problème plus complexe.

On suppose donnés les éléments de E sous forme d'un tableau de longueur n, formé en mettant les paires les unes à la suite des autres, triées dans l'ordre croissant de leur plus petit élément, et en ordonnant chaque paire de façon décroissante.

Par exemple (voir Figure 1), si l'ensemble des paires est :

$$\{1,6\}\ \{2,12\}\ \{3,5\}\ \{4,8\}\ \{7,11\}\ \{9,10\}$$

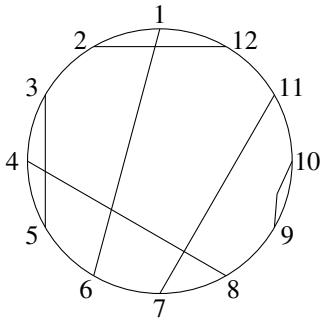

FIGURE 1. Un ensemble de cordes qui se coupent

le tableau représentant ces paires sera :

$$t = 6,1,12,2,5,3,8,4,11,7,10,9$$

On peut obtenir ce tableau en temps linéaire à partir des cordes données dans un ordre quelconque par une procédure simple, laissée à la sagacité du lecteur.

On note que l'exemple donné plus haut n'est pas réalisable sans croisement car, par exemple, les cordes $\{1,6\}$ et $\{2,12\}$ se coupent et la corde $\{4,8\}$ coupe trois autres cordes.

L'Algorithme 1 utilise une pile contenant des entiers. Ceux-ci doivent sortir de la pile dans l'ordre croissant de 1 à n.

À chaque étape, pour $i = 1,\ldots,n$, on effectue une itération qui supprime le haut de la pile s'il est égal au dernier élément sorti de la pile augmenté de 1 ; ensuite, on compare le nombre t_i avec le nombre qui se trouve au sommet de la pile, que l'on notera a dans la suite. Si la pile est vide, ou si $t_i < a$, on empile t_i ; sinon on peut montrer qu'il y a alors un croisement entre la corde dont une extrémité est a et celle dont une extrémité est t_i. En effet, on constate d'abord que la règle d'empilement implique la croissance des entiers dans la pile depuis le haut jusqu'au fond de la pile. Ainsi les t_k pour les valeurs de k paires font une entrée dans la pile suivie immédiatement d'une sortie. En posant $a = t_j$ on constate que les deux cordes considérées sont $c_1 = \{t_{j+1}, t_j\}$ et $c_2 = \{t_{i+1}, t_i\}$. On a ainsi $t_{j+1} < t_{i+1}$ car la corde c_1 apparaît avant c_2 dans t ; de plus $t_{i+1} < t_j$, car lorsque t_k se présente, avec k pair, tous les nombres qui se trouvent dans la pile sont supérieurs à t_k. De ce fait :

$$t_{j+1} < t_{i+1} < t_j < t_i$$

ce qui implique que c_1 et c_2 se croisent.

Données : Un ensemble de cordes représenté par un tableau t les contenant sous la forme décrite plus haut, P est une pile d'entiers supposée contenir initialement un entier plus grand que n de façon à éviter les accès à la pile vide.
Résultat : Vrai si aucune corde n'en coupe une autre.

```
dernierSorti ← 0;
pour i = 1,2,...,n faire
    tant que (Top(P) = dernierSorti + 1) faire
        Depiler(P) ;
        dernierSorti ← dernierSorti + 1
    fin
    si (non vide(P) et Top(P) < t[i]) alors
        return false
    sinon
        Empiler(P,t[i])
fin
return true
```

Algorithme 1 : Vérifier qu'un ensemble de cordes ne se coupent pas

On peut remarquer que la suite des nombres dans le tableau t est une permutation des entiers $1,2,\ldots,n$ et que l'algorithme teste si cette permutation peut être triée en utilisant une pile. En fait, on peut se contenter de vérifier que le tableau obtenu en ne retenant qu'un entier sur deux de t, ceux en position impaire, est bien triable par une pile. On peut aussi montrer que le nombre de permutations triables par une pile est égal au nombre d'ensembles de cordes sans croisement. Il se trouve que c'est la suite des *nombres de Catalan* $(1,2,5,14,42,132,\cdots)$, bien connue en combinatoire énumérative.

Une version moins simple

On considère maintenant une version du problème dont l'énoncé paraît semblable au précédent mais dont la solution est nettement moins simple. Elle fait intervenir un concept important introduit par R. E. Tarjan pour traiter le cas des graphes planaires quelconques. La donnée est la même que pour le cas précédent mais on autorise toutefois les cordes à être dessinées à l'intérieur ou à l'extérieur du cercle (voir l'exemple donné en Figure 2). Avec l'interprétation précédente, cela revient à tester si le tableau t peut être trié à l'aide de *deux* piles, l'une triant les cordes mises à l'intérieur l'autre celles à l'extérieur.

Une façon de faire consiste à construire un *graphe des croisements* dont les sommets correspondent aux cordes et tel que deux sommets sont reliés par une arête si

les cordes correspondantes se coupent lorsqu'elles sont mises du même coté, c'est-à-dire si elles sont de la forme $\{x_i, x_j\}$ et $\{x_k, x_\ell\}$ avec $i < k < j < \ell$. Tester si on peut représenter les cordes sans croisement revient alors à tester si les sommets du graphe ainsi formé sont coloriables en deux couleurs de façon telle que deux sommets reliés par une arête soient toujours de couleurs différentes. Ceci donne un algorithme dont le nombre d'opérations est linéaire en le nombre d'arêtes du graphe des croisements et qui semble donc efficace. Toutefois la construction de ce graphe nécessite l'examen de toutes les paires de cordes ce qui est de l'ordre de n^2. R. E. Tarjan a mis au point dans sa thèse une structure de données — qu'il a ensuite reprise de façon plus claire avec P. Rosenstiehl dans [11] — pour obtenir un algorithme en temps linéaire permettant de tester la bi-coloriabilité du graphe des croisements (voir Algorithme 2).

Cette structure fait intervenir une pile dont les cellules sont des couples de piles d'entiers : une pile gauche et une pile droite. Les opérations définies sur les cellules sont les suivantes :

— À partir d'un entier on peut **créer** une cellule dont la pile gauche contient cet entier et la pile droite est vide.
— On peut **échanger** la pile gauche et la pile droite d'une cellule.
— On peut **concaténer** deux cellules en mettant l'une à la suite de l'autre les deux piles gauches et les deux piles droites.
— On peut **accéder** à la pile gauche $cell.G$ ou à la pile droite $cell.D$ de la cellule $cell$ et y effectuer les opérations classiques sur les piles d'entiers.

Sur les piles de cellules on dispose des opérations courantes sur les piles :

— **Tester** si la pile est **vide**, **Empiler** une cellule en haut de la pile, **Dépiler** la cellule du haut de la pile.
— **Accéder** à la cellule en haut d'une pile P de deux façons différentes, soit par $Top(P)$ pour obtenir la cellule entière, soit par $TopG(P)$ (ou $TopD(P)$) pour obtenir l'entier qui se trouve en haut de la pile gauche (ou de la pile droite) de cette cellule.

On peut considérer ici la vérification de la réalisabilité comme la possibilité de trier la permutation représentée par le tableau t à l'aide de deux piles en parallèle : à chaque étape de l'algorithme on doit choisir de dépiler l'une des deux piles ou d'empiler l'élément suivant de la permutation sur l'une ou l'autre des piles.

En fait, il s'agit de représenter les possibilités de choix pour l'algorithme d'utiliser l'une ou l'autre des piles à sa disposition par cette structure de « pile de bi-piles » qui remet à plus tard le choix de la pile sur laquelle on empile un élément. Le choix est effectué lorsque l'on trouve une incompatibilité entre deux cordes qui se coupent.

L'Algorithme 2 permet de tester si une permutation quelconque peut être triée par deux piles, dans le cas où chaque élément ne peut être inséré que dans une seule des

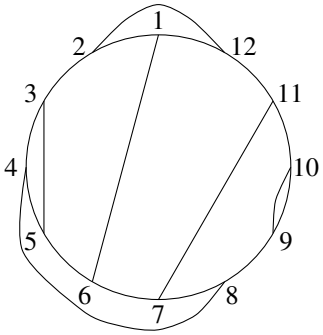

FIGURE 2. Un ensemble de cordes qui peut être représenté en interne et externe.

deux piles, au choix. Le nombre de ces permutations a été évalué de façon précise très récemment par Michael Albert et Mireille Bousquet-Mélou [1].

L'algorithme, dans le cas général

L'algorithme de planarité présenté dans la thèse de R. E. Tarjan — puis sous une forme simplifiée dans [6] —, utilise les piles de bi-piles dans sa dernière phase. Dans la première phase il s'agit de construire un arbre recouvrant [2] obtenu par une recherche en profondeur. Un tel arbre est également appelé *arbre de Trémaux*. En effet, ce bel algorithmique récursif a été pour la première fois proposé par Charles Trémaux (1859–1882), polytechnicien de la promotion 1876 et ingénieur des télégraphes, répondant à une question d'Édouard Lucas (1842–1891), professeur de Mathématiques en classe de spéciales au Lycée Saint-Louis, connu pour ses travaux en théorie des nombres et son ouvrage *Récréations Mathématiques* [9].

Il s'agissait de donner un algorithme précis permettant de sortir d'un labyrinthe, ce que l'on peut traduire par trouver un chemin qui visite toutes les arêtes d'un graphe. Édouard Lucas cite dans son ouvrage (voir page 47) une solution à son problème proposée par Charles Trémaux. Cet ouvrage est disponible sur le site Gallica de la Bibliothèque Nationale de France [3].

Plusieurs chercheurs, dont R. E. Tarjan, ont retrouvé l'algorithme proposé par Trémaux. Le mérite de R. E. Tarjan est d'avoir utilisé la numérotation des sommets obtenue dans l'ordre de ce parcours pour répondre à d'autres questions algorithmiques sur les graphes.

2. Un arbre recouvrant est un arbre contenant tous les sommets du graphe et obtenu en sélectionnant un sous-ensemble de ses arêtes.

3. http://gallica.bnf.fr/ark:/12148/bpt6k3943s

Données : Un ensemble de cordes représentées par un tableau t les contenant sous la forme décrite plus haut, P est une pile de cellules supposée contenir initialement une cellule avec deux piles contenant chacune un entier plus grand que n de façon à éviter les accès à la pile vide.
Résultat : Vrai si on peut donner une représentation planaire des cordes de manière interne ou externe au cercle, Faux sinon.

```
doitSortir ← 1;
pour i = 1,2,...,n faire
    tant que (TopG(P) = doitSortir ou TopD(P) = doitSortir) faire
        Depiler(Top(P).G) ou Depiler(Top(P).D) suivant le cas;
        si (estVide(Top(P).G) et estVide(Top(P).D) alors
            Depiler(P);
        doitSortir ← doitSortir + 1
    fin
    cell1 ← Top(P);
    cell2 ← CreerCellule(t[i]);
    tant que (t[i] > Top(cell1.G) ou t[i] > Top(cell1.D)) faire
        si t[i] < Top(cell1.D) alors
            Echanger(cell1) ;
        si t[i] > Top(cell1.G) alors
            return false
        sinon
            cell2 ← Concatener(cell2, cell1);
            Depiler(P);
            cell1 ← Top(P);
        fin
    fin
    Empiler(P, cell2);
fin
return true
```

Algorithme 2 : Vérifier qu'un ensemble de cordes peut être réalisé en interne et en externe

Son idée fondamentale a été d'associer à chaque sommet x_i une valeur qu'il a appelée « point le plus bas » (*lowpoint*) : elle désigne le numéro du sommet de plus faible numéro dans l'arbre de Trémaux qui peut être atteint par une seule arête en dehors de cet arbre depuis un sommet situé dans le sous-arbre ayant pour racine x_i. Ce nouveau paramètre a fait merveille dans d'autres algorithmes sur les graphes comme la recherche des composantes biconnexes ou triconnexes, le test de forte connexité pour les graphes orientés et d'autres applications.

Une deuxième phase consiste à décomposer le graphe en un ensemble de chemins, il y en a autant que d'arêtes en dehors de l'arbre. Ces chemins jouent ensuite

un rôle similaire aux cordes que nous avons considérées dans les problèmes étudiés plus haut. Il s'agit alors de savoir pour chaque chemin s'il doit être plongé à l'intérieur ou à l'extérieur d'un cycle principal de la construction. Cette décision doit être prise en fonction d'incompatibilités qui ont un statut très voisin du graphe construit sur les cordes. L'utilisation d'une pile de bi-piles est alors cruciale pour obtenir un algorithme qui donne un résultat en temps linéaire.

Il serait bien long de donner la totalité de l'algorithme dans ces lignes. J'ai choisi de mettre en lumière deux points précis qui me paraissent être les points clefs de l'algorithme. Les lecteurs souhaitant connaître les détails complets peuvent consulter l'article original de Hopcroft et Tarjan [6] ou une tentative de présentation plus intuitive par William Kocay [7].

La fonction « point le plus bas » de R. E. Tarjan

L'algorithme de parcours en profondeur (*DFS*) ou de Trémaux consiste, lorsque l'on atteint un sommet au cours du parcours, à trouver un sommet voisin non encore visité ; si tous les voisins ont déjà été visités il faut alors retourner en arrière par l'arête qui a permis d'atteindre ce sommet pour la première fois et recommencer l'opération. À chaque fois que l'on atteint un nouveau sommet on le numérote par l'entier le plus petit non encore utilisé. L'ensemble des arêtes qui permettent d'atteindre un nouveau sommet forment alors un arbre et les arêtes en dehors sont appelées *arêtes de retour*. En orientant les arêtes de l'arbre du sommet de numéro le plus petit vers celui de numéro le plus grand et les arêtes de retour du sommet de numéro le plus grand vers le sommet de numéro le plus petit on obtient ce que Tarjan appelle un *palmier*.

La fonction *lwpt* (point le plus bas) associe au sommet x_i le numéro le plus petit du sommet qui peut être atteint par un chemin orienté composé d'une suite d'arêtes de l'arbre (dont les numéros vont donc en croissant) et d'un seul arc de retour (qui permet de faire décroître le numéro). La détermination des valeurs de *lwpt* peut s'obtenir en ajoutant quelques instructions à l'algorithme de parcours *DFS* qui calcule l'arbre recouvrant et qui numérote aussi les sommets du graphe dans l'ordre du parcours (voir Algorithme 3).

L'entier k, les tableaux d'entiers $pere, num, lwpt$ sont des variables globales, *num* est un tableau initialisé à 0 pour tous les sommets du graphe, qui est ensuite modifié lorsqu'on atteint le sommet pour la première fois. Un sommet x_0 du graphe est distingué comme la racine de l'arbre recouvrant à construire, la fonction est appelée par $LWPT(x_0)$. À la suite du calcul, les *num* et *lwpt* sont obtenus pour tous les sommets. L'arbre est fourni par la fonction *pere*.

Données : Un graphe donné par les listes des voisins de ses sommets.
Résultat : Détermination de l'arbre recouvrant par la fonction *pere*, numérotation des sommets dans l'ordre du parcours en profondeur, et de la fonction $lowpt$.

```
k ← 1;
pour Tout i = 1...n faire
    num[x_i] ← 0
fin

DFSLWPT(x) :
lwpt[x] ← num[x];
pour Tout voisin y de x faire
    si num[y] ≠ 0 alors
        lwpt[x] ← Min(num[y], lwpt[x])
    sinon
        pere[y] ← x;
        num[y] ← k;
        k ← k + 1;
        DFSLWPT(y);
        lwpt[x] ← Min(lwpt[y], lwpt[x])
    fin
fin
```

Algorithme 3 : La fonction récursive $LWPT(x)$

Cette fonction présente plusieurs intérêts en dehors de l'algorithme de planarité. Par exemple, elle permet de savoir si un sommet x d'un graphe est un point d'articulation, c'est-à-dire si la suppression de x et des arêtes qui lui sont incidentes déconnecte le graphe. En effet, il suffit de vérifier que $lwpt(y) = x$ où y est l'un des fils de x dans un arbre de Trémaux. Sur l'exemple donné en Figure 3, un graphe est dessiné avec un arbre de Trémaux en traits épais, les sommets sont numérotés dans l'ordre de parcours de l'arbre, et les valeurs de $lwpt$ sont données par :

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$lwpt[i]$	1	1	1	3	3	4	4	6	7	1	1	2	12	12

On remarque que 12 est un point d'articulation : en effet, son fils 13 vérifie $lwpt[13] = 12$. Il en est de même de 3 car $lwpt[4] = 3$.

Pour ce qui est de l'algorithme de planarité, on peut considérer que cette fonction permet de remplacer chaque chemin composé d'une suite d'arêtes de l'arbre et d'une seule arête de retour par une corde sur un cercle formé par un cycle du graphe ; on se ramène ainsi à un problème voisin de ceux qui ont fait l'objet des deux premières sections de ce texte.

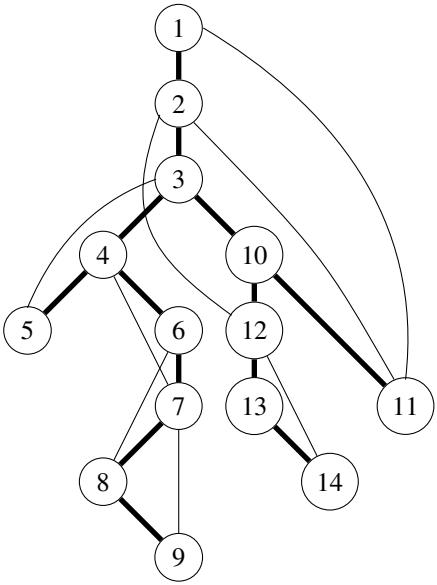

FIGURE 3. Un arbre de Trémaux, les arêtes de retour sont en trait fin

Histoire de l'algorithme

Cet algorithme a été proposé par R. E. Tarjan dans sa thèse de Ph.D. à l'Université de Stanford, sous la direction de Robert W. Floyd et Donald Knuth. Cette thèse fait 133 pages de texte et contient le texte du programme qui implante l'algorithme. Il est écrit en Algol W et fait moins de 1000 lignes de code et commentaires. Le programme avait été testé sur des graphes planaires allant jusqu'à 900 sommets et 2500 arêtes. Il prenait une dizaine de secondes sur un IBM 360/67 pour vérifier la planarité de tels graphes.

Après sa thèse, R. E. Tarjan a été nommé professeur à Cornell où il a repris l'algorithme de sa thèse avec John Hopcroft, professeur à Cornell. Ils en ont donné une version plus courte et plus lisible (voir [6]). Par la suite, de nombreux chercheurs ont implanté cet algorithme en l'améliorant, c'est le cas en particulier de Pierre Rosenstiehl et ses élèves Hubert de Fraysseix et Patrice Ossona de Mendez.

Les travaux de Hopcroft et Tarjan sur les algorithmes et les structures de données ont été couronnés par le prix Turing en 1986, la plus haute reconnaissance pour des chercheurs en informatique. Les travaux de R. E. Tarjan, qui est actuellement professeur à l'Université de Princeton, ont aussi été reconnus par de nombreux autres prix et distinctions.

En ce qui me concerne, j'ai connu cet algorithme au début de l'année 1973. Je venais d'être nommé à Bordeaux et j'ai été destinataire d'un exemplaire de la thèse de R. E. Tarjan grâce à Jean Vuillemin qui avait fréquenté pendant plusieurs années l'Université de Stanford dans une équipe de chercheurs gravitant autour de Donald Knuth. J'avais tout de suite proposé comme sujet de projet, à un groupe d'étudiants de Maîtrise d'Informatique, la réalisation de cet algorithme en Algol sur les machines IBM du Centre de calcul de l'Université. Un jeune garçon brillant avait réussi à conduire ce groupe qui a mené à terme ce projet. Il s'agit de Dominique Gouyou-Beauchamps, qui s'est illustré par la suite en Combinatoire et Algorithmique à Bordeaux puis à Orsay.

Références

[1] M. Albert, M. Bousquet-Mélou. Permutations sortable by two stacks in parallel and quarter plane walks. **arXiv** :1312.4487, 35p. (2013).

[2] L. Auslander, S. Parter. On Imbedding Graphs in the Sphere. *J. Math. Mechanics* **10**, 517–523 (1961).

[3] G. Demoucron, Y. Malgrange, R. Pertuiset. Graphes Planaires : reconnaissance et construction de représentations. *Revue Française de Rech. Op.* **8**, 33–47 (1964).

[4] H. de Fraysseix, P. Ossona de Mendez. Trémaux trees and planarity. *Eur. J. Combinatorics* **33**, 279–293 (2012).

[5] H. de Fraysseix, P. Rosenstiehl. A characterization of planar graphs by Trémaux orders. *Combinatorica* **5**, 127–135 (1985).

[6] J. Hopcroft, R. E. Tarjan. Efficient planarity testing. *J. Assoc. Comp. Mach.* 21, 10–22 (1974).

[7] W. Kocay. The Hopcroft-Tarjan planarity algorithm (1993). `http://www.combinatorialmath.ca/G&G/articles/planarity.pdf`

[8] A. Lempel, S. Even, I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs, International Symposium, Rome*, Gordon and Breach, New York (1967).

[9] E. Lucas, *Récréations Mathématiques*, 2e édition, Librairie Albert Blanchard Paris (1891). Nouveau tirage 1992.

[10] R. E. Tarjan. *An efficient planarity algorithm*. Ph.D. Thesis, Stanford University (1972).

[11] P. Rosenstiehl, R. E. Tarjan. Gauss codes, planar Hamiltonian graphs and stack-sortable permutations. *J. Algorithms* **5**, 375–390 (1984).