|  |  |
| --- | --- |
|  | Projet de théorie des graphes |
|  |  |
| Berthomier Léa Guillet Valentin | Recherche des tours d’Euler dans un graphe |
|  | Le but de notre projet est de construire un algorithme capable d’indiquer si un graphe donné contient un tour d’Euler et, auquel cas, de le trouver. |

Projet de théorie des graphes

Recherche des tours d’Euler dans un graphe

# Principe theorique

Afin de savoir si un graphe contient un tour d’Euler, nous nous sommes basés sur le théorème d’Euler vu en cours :

Soit G un graphe connexe et sans boucle.

Alors G possède un tour d’Euler si et seulement si tous les sommets de G sont de degré pair.

Notre algorithme de reconnaissance de graphe eulérien consiste donc uniquement à examiner chaque sommet du graphe et vérifier qu’il est bien de degré pair.

En ce qui concerne la recherche d’un tour d’Euler dans un graphe, nous avons utilisé la démonstration de ce théorème qui donne une méthode explicite pour en trouver un.

*Démontrons par double implication ce théorème :*

*Soit G un graphe connexe et sans boucle possédant un tour d’Euler, soit v un sommet de G.*

*Alors lors du parcours de ce tour, à chaque passage par le sommet v, le tour arrive et repart de ce sommet par deux arêtes différentes.*

*Donc le sommet x est de degré pair.*

*Par récurrence forte sur le nombre de sommets n d’un graphe :*

*Initialisation : cas  : le graphe à un sommet ne contient pas d’arêtes, il est donc Eulérien*

*Itération : soit on suppose le résultat vrai pour tout .*

*Soit G un graphe possédant n+1 sommets.*

*Comme tous les sommets de G sont de degré pair, on a montré dans le cours que G possède un cycle, que l’on note C.*

*On considère alors le graphe H égal à G auquel on a retiré les arêtes présentes dans C. Chaque composante connexe de H contient alors moins de n sommets et tous ces sommets sont de degré pair puisque l’on a retiré un nombre pair d’arêtes sur chaque sommet, on peut donc appliquer l’hypothèse de récurrence, et former un tour d’Euler sur chacune de ces composantes connexes.*

*Finalement, on construit le tour suivant : on parcourt C, puis à chaque sommet appartenant à une composante connexe de H, on substitue ce sommet par un tour d’Euler de cette composante connexe.*

*Ainsi, le tour obtenu est bien un tour d’Euler dans G tout entier.*

# Notre Algorithme de recherche

La méthode présentée précédemment nous permet donc de trouver un tour d’Euler. Toutefois, il faut pour cela être capable de trouver les cycles dans le graphe donné. Nous avons donc mis au point la méthode suivante.

## Rechercher un cycle

Le principe de cet algorithme est de partir d’un point v0 du graphe et d’inspecter tous les voisins de v0, de plus en plus loin, jusqu’à retomber sur un point déjà inspecté.

*Soit G un graphe à n sommet. Soit S l’ensemble des sommets inspectés, vide initialement. Soit A={v0}, contenant un unique sommet de départ, et qui contiendra les sommets à analyser. On donne à chaque sommet v du graphe un attribut chemin C qui est une liste, initialement vide, contenant les sommets composants un chemin de v0 à v*.

*Pour tout : //POURQUOI ?*

*Extraire un sommet v de A. Ajouter v à S.*

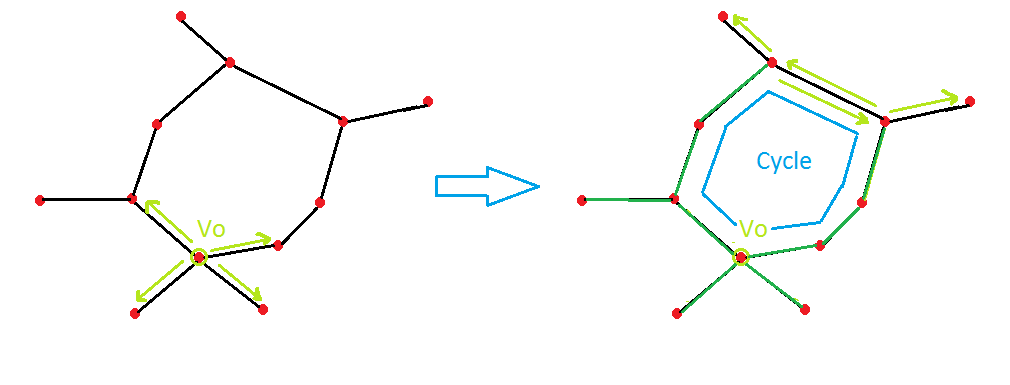
*Pour tout sommet w voisin de v :*

*- Si w n’appartient pas à S ou si v et w sont sur deux branches différentes du cycle partant de v0, i.e. C(v)[1] != C(w)[1], alors :*

*- Si w a déjà été visité, i.e. C(w) !=[] : un cycle a été trouvé. On renvoie la concaténation de C(v), C(w), et v-w, qui forme le cycle.*

*- Sinon : C(w) = C(v) + v*

* *On ajoute w à A pour analyser ensuite ses voisins.*



## Rechercher un tour d’Euler

Nous utilisons ensuite la recherche récursive de cycles afin d’obtenir un tour d’Euler. Cependant, il reste à s’assurer que, dans la recherche d’un nouveau cycle dans une composante du graphe, une des arrêtes du cycle n’est pas déjà utilisée dans le tour d’Euler, sinon nous aurions des arrêtes utilisées deux fois. Pour cela, nous avons utilisé une variable globale *edges* qui est une liste de booléens indiquant si l’arrête a déjà été utilisée. Ainsi dans la recherche d’un cycle nous ne considérons pas les voisins w qui sont reliés à v par une arrête déjà utilisée.

# EN pratique

## Afin de tester notre algorithme

Nous avions pour but de réaliser une interface graphique représentant un graphe généré aléatoirement et un de ses tours d’Euler.

**Méthode de génération aléatoire de graphes :**

*Soit n un entier et p Є [0 ; 1]. On place n sommets sur un cercle. Pour tout couple (v,w) de sommets du graphes, tels que v !=w, on trace une arrête entre v et w avec la probabilité p. La représentation encercle permet de ne pas avoir d’arrêtes qui se superposent.*

Afin de pouvoir tester notre algorithme qui vérifie si un graphe est Eulérien ou non, il nous faut d’abord s’assurer que le graphe en question est connexe. En effet, la méthode qui permet de vérifier qu’un graphe est Eulérien est simple, mais elle suppose que le graphe soit connexe, sinon on ne peut trouver de tour d’Euler que sur une seule des composantes. Nous avons donc dû écrire une fonction qui vérifie la connexité d’un graphe.

**Méthode de vérification de la connexité :**

(Dans notre programme, le degré d’un sommet est l’un de ses attributs.)

*Soit G un graphe à n sommets. Soit S l’ensemble des sommets inspectés, vide initialement. Soit A={v0}, contenant un unique sommet de départ, et qui contiendra les sommets à analyser.*

*Si un vertex est de degré 0 : retourner False.*

*Tant que A !={ } et Card( S ) != n :*

*Soit v Є A. On ajoute v à S. Soit w un voisin de v. Si w n’appartient ni à A, ni à S, alors on ajoute w à A.*

*Si Card( S ) = n, on retourne True, sinon False.*

Nous pouvons donc maintenant déterminer si n’importe quel graphe donné est Eulérien ou non. Toutefois, la probabilité d’obtenir un graphe Eulérien aléatoirement étant infime (environ patate chances sur 10000), nous avons besoin d’en fabriquer afin de tester notre programme.

**Méthode pour obtenir un graphe Eulérien :**

*//good luck here*

## Interface graphique

// Description, intérêt, difficulté,… tant que ça fait des lignes en plus, écrit ce que tu veux !

## Limites / fiabilité / complexité de l’algorithme

// *Summons the legendary Golden Pipeau*

*Représentation graphique dégueu au-delà de 30 sommets et p > 0.5*

FindCycle en O(n²), findEulérian en O(max(Card E, n²)), avec Card E = n(n-1)/2 \* p , donc algo en **O(n²)** ??? complexité makeEulerian ???