**Devoir #1**

PIF1005

Scott Le Clair

Université du Québec à Trois-Rivières

Hiver 2023

Table des matières

[Introduction 1](#_Toc132048784)

[Les structures des données 2](#_Toc132048785)

[Les algorithmes 3](#_Toc132048786)

[Les algorithmes identiques entre GrapheA.java et GrapheI.java 3](#_Toc132048787)

[évaluation () – Dirige les autres méthodes pour créer les données à afficher à la console 3](#_Toc132048788)

[contientCycle() – Trouve si un cycle existe dans le graphe 5](#_Toc132048789)

[contientCycleEulérien() – Trouve s’il existe un cycle eulérien dans le graphe 6](#_Toc132048790)

[contientChaineEulérien() – Trouve s’il existe une chaîne eulérien dans le graphe 7](#_Toc132048791)

[estPlanaire() – Monte si le graphe est planaire 8](#_Toc132048792)

[identifieSousGraphes() – Traverse le graphe plusieurs fois pour trouver des sous graphes 10](#_Toc132048793)

[trouveTricycle() – Trouve s’il existe un tricycle dans le graphe 11](#_Toc132048794)

[trouveDegres() – Identifie chaque sommet et leur degré 12](#_Toc132048795)

[compteArcsEtSommetsSG() – Compte les arcs et sommets dans un sous graphe 13](#_Toc132048796)

[parcourirSimple() – Traverse le graphe, pour comparer les matrices 14](#_Toc132048797)

[parcourir() – Traverse le matrice, identifie tous les arcs et sommets 15](#_Toc132048798)

[Des méthodes unique/uniquement implémentées en GrapheA.java 16](#_Toc132048799)

[arcsIncidents() – Identifie les arcs incident à un sommet 16](#_Toc132048800)

[chercheIncidence() – Identifie un sommet par un sommet adjacent et leur arc 17](#_Toc132048801)

[identifieArcs() – Déduit tous les arcs dans le graphe 18](#_Toc132048802)

[compteArcsEtSommets() – Compte les arcs et sommets dans le graphe 19](#_Toc132048803)

[sommetsAdjacents () - Identifie tous les sommets adjacent à un sommet 20](#_Toc132048804)

[Des méthodes unique/uniquement implémentées en GrapheI.java 21](#_Toc132048805)

[arcsIncidents() – Identifie les arcs incident à un sommet 21](#_Toc132048806)

[chercheIncidence() – Identifie un sommet par un sommet adjacent et leur arc 22](#_Toc132048807)

[compteArcsEtSommets() – Compte les arcs et sommets dans le graphe 23](#_Toc132048808)

[Les entrées choisis (voir Main.java) 24](#_Toc132048809)

[Les résultats des traitements 28](#_Toc132048810)

[Pensées concernant le choix entre matrice d’adjacence et incidence pour représenter un graphe 32](#_Toc132048811)

# Introduction

Ce document contient des descriptions et des explications pour les choix de conception prises pour le programme Java de devoir 1, qui évalue des graphes simples. Le but de ce document est de mettre en contexte le programme associé.

# Les structures des données

Pour commencer, le programme fait le traitement d’un graphe à l’aide d’un des deux classes : un pour les graphes représentés par un matrice d’adjacence, et l’autre pour les graphes représentés par un matrice d’incidence. Le développeur a fait ce choix, parce que le traitement des matrices diffèrent d’une manière importante.

Le développeur a choisi **un tableau à deux dimensions** pour représenter une matrice, puisque la matrice ne changera pas pendant le traitement d’un graphe et le développeur voulait mesurer exactement la quantité d’itérations et affectations fait pendant le programme traverse un graphe.

Au début, les données étaient tous structurées sous forme de tableaux, ce qui a marchais assez bien, mais le manque des méthodes de **ArrayList** a augmenté considérément le temps de développement. Plus important, il y avait une situation où le développeur n’a pas pu identifier la taille potentielle d’un tableau.

Cette situation a été quand le développeur voulait représenter les arcs que l’algorithme « parcourir() » (recherche en profondeur) a pris lors de son recherche (« arcsD », informellement appelés arcs directs) et les arcs de redondance (« arcsA », informellement appelés arcs arrières). Donc, ce fait a motivé le développeur à changer tous les tableaux en ArrayLists (sauf que les matrices).

Une chose importante à mentionner est le fait que dans « GrapheA.java » presque tous les ArrayLists concernant les arcs, entreposent des indices pour un ArrayList « Maitre » (retourné par la méthode GrapheA.identifieArcs()) qui contient les arcs sous forme d’adjacence unique (il n’y a pas de doublon comme [1,0] et [0,1]). Le développeur a créé ce méthode parce qu’il faut déterminer les arcs pour plusieurs traitements dans le programme. Plus que la moitié des méthodes dans GrapheA font référence à identifieArcs().

Finalement, un **HashMap<String, Integer>** a été utilisé pour représenter le nom de chaque sommet et son propre degré. Un HashMap est utile pour associer deux valeurs des types différents, et le développeur voulait utiliser une chaîne (e.g. « S0 », « S1 », « S2 ») pour représenter chaque sommet, qui a aussi un degré associé (sous forme d’entier). Les sous graphes sont représenter par un **HashMap <String, ArrayList<Integer>>**, afin d’associer un identificateur à une liste des sommets.

# Les algorithmes

Le programme contient 3 fichiers :

* **Main.java** – le fichier principale qui contient le méthode pilot et l’entrée des données de graphe sous forme de matrice
* **GrapheI.java** – le ficher qui représente et traite un graphe par sa matrice d’incidence
* **GrapheA.java** – le ficher qui représente et traite un graphe par sa matrice d’adjacence

Les algorithme parcourirSimple() et tous les méthodes sur lesquelles parcourirSimple() dépend, ils tiennent compte du nombre d’itérations et affectations. Le programme traverse les matrices d’incidence et d’adjacence différemment.

## Les algorithmes identiques entre GrapheA.java et GrapheI.java

### évaluation () – Dirige les autres méthodes pour créer les données à afficher à la console

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Entrée :

sommet (entier) - > représente le sommet où commencer le recherche en profondeur

Procès :

Traite le graphe, prépare l’affichage

Sortie :

Affiche les résultats à la console à l’aide de System.out.print().

### contientCycle() – Trouve si un cycle existe dans le graphe

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Identifie les parties non connexe à l’aide de identifieSousGraphes(). Pour chaque sous graphe, traverse-le à l’aide de parcourir(). Si un sous graphe contient un arc dans arcA (arcs de redondance), le graphe contient un cycle.

Sortie :

Vrai si le graphe contient un cycle, sinon faux

### contientCycleEulérien() – Trouve s’il existe un cycle eulérien dans le graphe

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Identifie s’ils existent des parties non connexe à l’aide de identifieSousGraphes(). Assure que le graphe est connexe. Vérifie chaque pair de sommet et degré dans « degres ».

Sortie :

Vrai si tous les sommets a un degré pair, sinon faux

### contientChaineEulérien() – Trouve s’il existe une chaîne eulérien dans le graphe

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Identifie s’ils existent des parties non connexe à l’aide de identifieSousGraphes(). Assure que le graphe est connexe. Vérifie chaque pair de sommet et degré dans « degres ». Tiens compte du nombre des sommets qui ont un degré impair.

Sortie :

Vrai si exactement deux sommets ont un degré impair, sinon faux

### estPlanaire() – Monte si le graphe est planaire

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Identifie les parties non connexe à l’aide de identifieSousGraphes(). Pour chaque sous graphe, traverse-le à l’aide de parcourir(). Pour chaque sous graphe, évalue le nombre de sommets, arcs et si un cycle de trois soit présent. Examine le graphe sous les conditions essentielles mais non suffisantes pour détecter les graphs K(5) et K(3,3).

Sortie :

Vrai s’il est possible que le graphe soit planaire, sinon faux (certainement impossible)

### identifieSousGraphes() – Traverse le graphe plusieurs fois pour trouver des sous graphes

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Pour chaque sommet, traverse le graphe. En traversant le graphe, trouve les sommets qui sont adjacent. Initialise une liste tampon. Ajoute les sommets dans une liste tampon. Si c’est le premier graphe évalué, ajoute-le automatiquement à « graphes ». Sinon, vérifie que la liste tampon contient des sommets qui ne sont pas déjà dans « graphes ». Ajoute la liste tampon dans « graphes » avec un clef, si le contenu de la liste tampon n’est pas déjà dans « graphes ».

Sortie :

« graphes », un HashMap de sous graphes identifiés par un clef

### trouveTricycle() – Trouve s’il existe un tricycle dans le graphe

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Pour chaque sommet (père) dans un graphe, trouves ses fils. Pour chaque fils, trouve ses frères. Pour chaque frère, vérifie qu’ils sont le frère est aussi un fils avec le père mentionné.

Sortie :

Vrai si un père a deux enfants qui sont adjacents, sinon faux

### trouveDegres() – Identifie chaque sommet et leur degré

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Initialize le HashMap qui contient les degrés. Pour chaque sommet, crée un clef. Pour chaque incidence ou adjacence dans la matrice, ajoute 1 au degré de ce sommet. Mets le clef et le sommet dans le HashMap de degrés.

Sortie :

Rien

### compteArcsEtSommetsSG() – Compte les arcs et sommets dans un sous graphe

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Pour chaque sommet et arc visité par identifieSousGraphes(), ajoute 1 à son compteur respectif.

Sortie :

Rien

### parcourirSimple() – Traverse le graphe, pour comparer l’efficacité des matrices

Une image contenant texte

Description générée automatiquement

Entrée :

sommet (int) -> le sommet du début de parcours

Procès :

Marque le premier sommet comme visité. Trouve ses sommets adjacents. Fait appel à lui-même. Compte chaque itération de boucle et chaque affectation de variable.

Sortie :

Rien

### parcourir() – Traverse le matrice, identifie tous les arcs et sommets

Une image contenant texte

Description générée automatiquement

Entrée :

sommet (int) -> le sommet du début de parcours

Procès :

Marque le premier sommet comme visité. Pour chaque arc incident à ce sommet, vérifie si l’arc à été exploré. Sinon, explore-le, et si le sommet à l’autre côté n’a pas été déjà visité, marque cet arc comme visité et fait la récursion. Si le sommet à l’autre côté a été déjà visité, marque cet arc comme visité et vérifie l’arc prochain.

Sortie :

Rien

## Des méthodes unique/uniquement implémentées en GrapheA.java

### arcsIncidents() – Identifie les arcs incident à un sommet

Une image contenant texte

Description générée automatiquement

Entrée :

sommet (int) -> le sommet pour lequel on veut trouver tous les arcs incidents

Procès :

Identifie tous les arcs du graphe par identifieArcs(). Pour chaque arc, si l’arc est incident au sommet, ajoute l’arc dans une liste.

Sortie :

Une ArrayList « arcs » qui contient tous les indices des arcs incidents au sommet (les indices sont significatifs pour la liste maître des arcs dans identifieArcs())

### chercheIncidence() – Identifie un sommet par un sommet adjacent et leur arc

Une image contenant texte

Description générée automatiquement

Entrée :

sommet (int) -> le sommet pour lequel on veut trouver un adjacence

arcId (int) -> un indice qui représente un arc dans identifieArcs()

Procès :

Identifie tous les arcs du graphe par identifieArcs(). Pour l’arc donné par l’indice « arcId », cherche le sommet qui n’est pas égale à « sommet » pour trouver le sommet adjacent.

Sortie :

« adjacent » (int) un sommet adjacent à « sommet » qui partage un arc spécifié par l’indice « arcId »

### identifieArcs() – Déduit tous les arcs dans le graphe

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Identifie les parties non connexe à l’aide de identifieSousGraphes(). Pour chaque sous graphe, traverse-le à l’aide de parcourir(). Si un sous graphe contient un arc dans arcA (arcs de redondance), le graphe contient un cycle.

Sortie :

Un ArrayList des Arraylists<Integer> « arcs », qui représente chaque arc dans le graphe par les deux sommets qui sont adjacent

### compteArcsEtSommets() – Compte les arcs et sommets dans le graphe

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

Parcours la matrice d’adjacence. Pour chaque rangé, ajoute 1 à « nbrSommets » (le nombre de sommets dans le graphe). Vérifie chaque valeur de la colonne. Si la valeur est égale à 1, incrémente nbrArcs (le nombre d’arcs dans le graphe) par un. Finalement, divise le nombre d’arcs par 2, parce que chaque adjacence entre deux sommets contribue à un arc.

Sortie :

Rien

### sommetsAdjacents () - Identifie tous les sommets adjacent à un sommet

Une image contenant texte

Description générée automatiquement

Entrée :

sommet (int) -> le sommet pour lequel on veut trouver tous les sommets adjacents

Procès :

Parcours le rangé de la matrice (spécifié par « sommet »). Si une valeur dans le rangé est égale à 1, ajoute le sommet dans une liste. Tiens compte de chaque itération et affectation.

Sortie :

Une ArrayListe<Integer> « sommets » avec tous les sommets qui sont adjacent à « sommet »

## Des méthodes unique/uniquement implémentées en GrapheI.java

### arcsIncidents() – Identifie les arcs incident à un sommet

Une image contenant texte

Description générée automatiquement

Entrée :

sommet (int) -> le sommet pour lequel on veut trouver tous les arcs incidents

Procès :

Parcours le rangé de la matrice (spécifié par « sommet »). Si une valeur dans le rangé est égale à 1, ajoute l’arc dans une liste. Tiens compte de chaque itération et affectation.

Sortie :

Une ArrayListe<Integer> « arcs » avec tous les arcs qui sont incident à « sommet »

### chercheIncidence() – Identifie un sommet par un sommet adjacent et leur arc

Une image contenant texte

Description générée automatiquement

Entrée :

sommet (int) -> le sommet pour lequel on veut trouver un adjacence

arc (int) -> un indice qui représente un arc dans la matrice

Procès :

Parcours les rangés dans la matrice d’incidence et vérifie la valeur positionnée dans le colonne « arc ». Le deuxième rangé qui contient la valeur de 1 dans le colonne « arc » est le sommet adjacent. Tiens compte de chaque itération et affectation.

Sortie :

« adjacent » (int) un sommet adjacent à « sommet » qui partage un arc spécifié par « arc »

### compteArcsEtSommets() – Compte les arcs et sommets dans le graphe

Une image contenant texte

Description générée automatiquement

Entrée :

Rien

Procès :

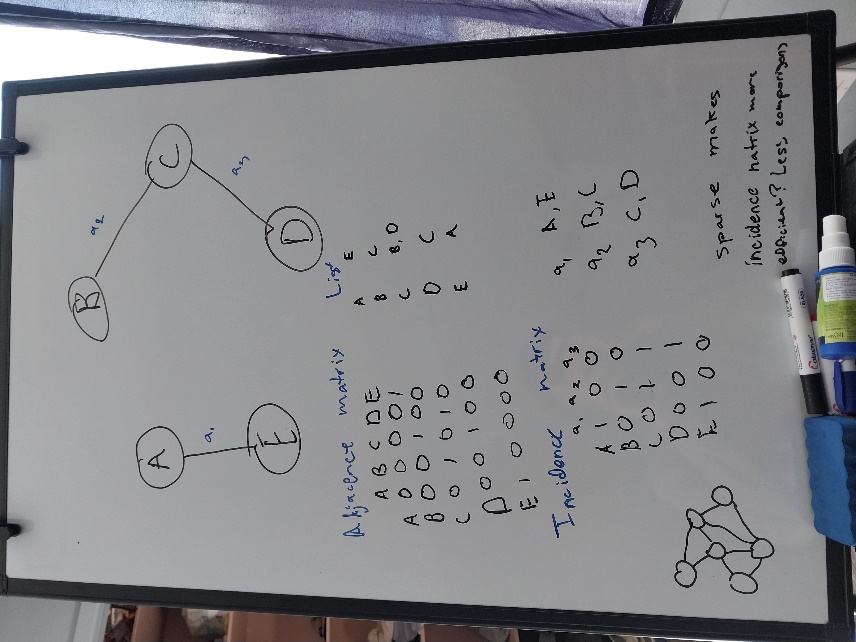
Pour chaque rangé dans la matrice d’incidence, incrémente « nbrSommets ». Pour chaque colonne, incrémente « nbrArcs ».

Sortie :

Rien

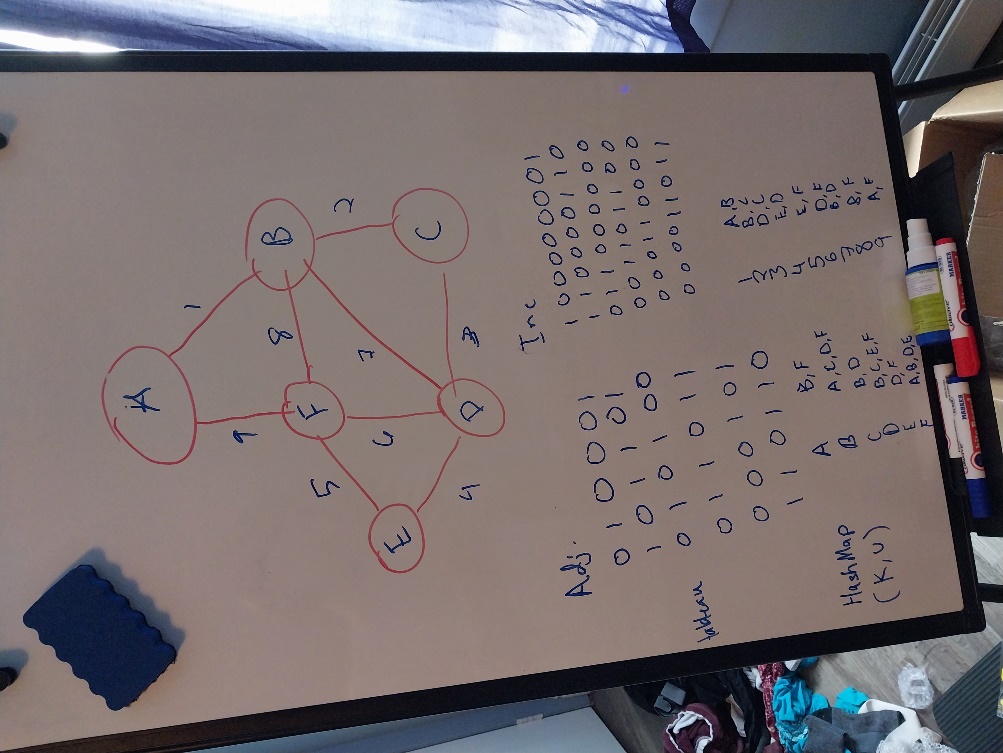
# Les entrées choisis (voir Main.java)

#### Graphe 1

 Une image contenant texte

Description générée automatiquement

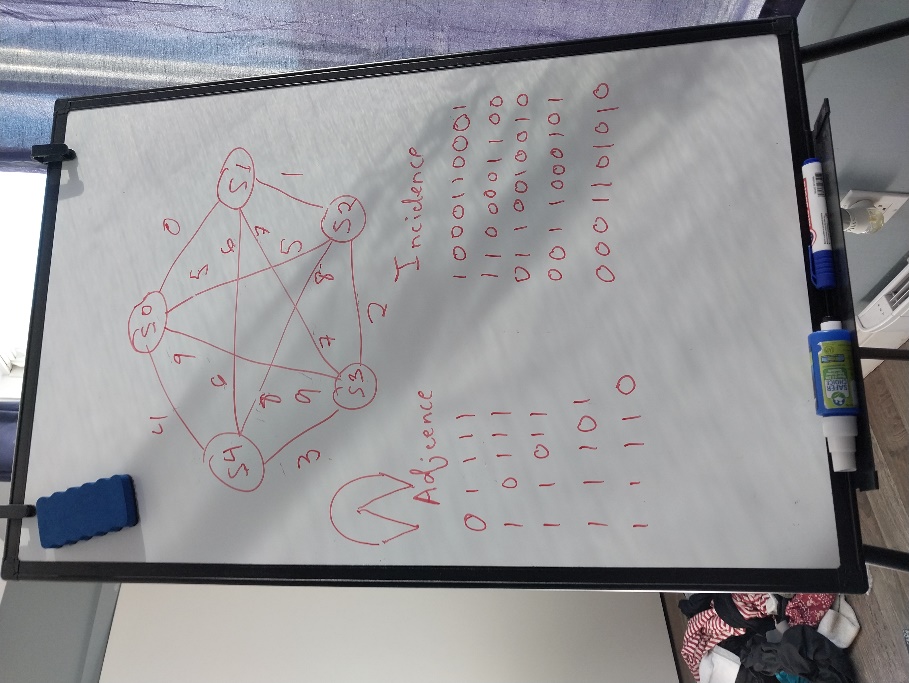
#### Graphe 2



Une image contenant texte

Description générée automatiquement

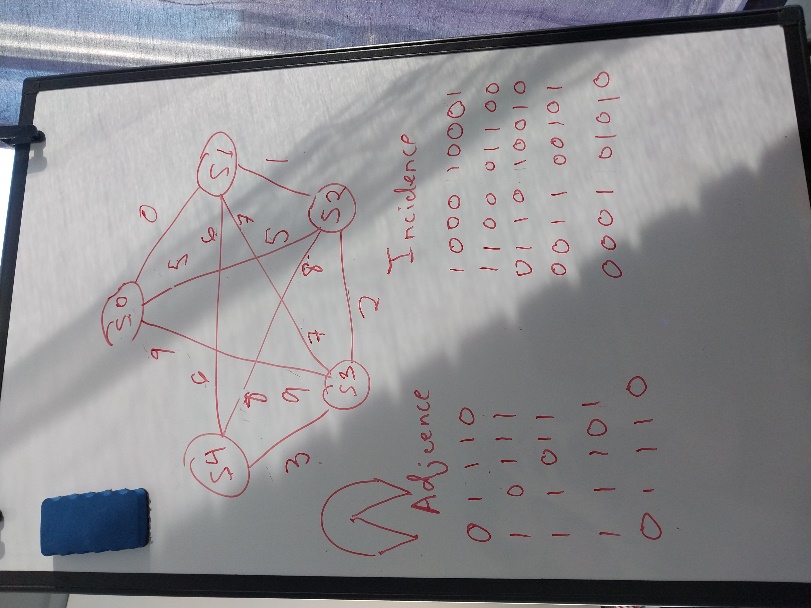
#### Graphe 3



Une image contenant texte

Description générée automatiquement

#### Graphe 4



\*Les arcs ne sont pas numérotés bien ci-dessus

Une image contenant texte

Description générée automatiquement

# Les résultats des traitements

#### Graphe 1

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

#### Graphe 2

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

#### Graphe 3

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

#### Graphe 4

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

# Pensées concernant le choix entre matrice d’adjacence et incidence pour représenter un graphe

Les matrices d'adjacence sont plus faciles à implémenter pour faire une traverse d’un graphe simple, mais n'indique pas directement les arcs. Dans un graphe simple, on peut déduire chaque arc par le degré de chaque sommet, mais dans un multigraphe, les matrices d'adjacence manquent les informations requises pour représenter tous les arcs dans un programme comme ceci. La matrice adjacence devient plus efficace dans cet algorithme (relative à la matrice d'incidence) quand le graphe a moins de sommets et plus d'arcs, jusqu'à un graphe complet.

Les matrices d'incidences sont plus difficiles à implémenter pour traiter des graphes simples. Mais l’implémentation de ce programme pour traiter des multigraphes pourrait être beaucoup plus facile en représentant les graphes avec des matrices d’incidence.

Il faut parcourir les matrices d’incidence plus souvent qu'avec les matrices d'adjacence pour traverser un graph simple. Pour chercher une adjacence dans la matrice d’incidence, il faut rechercher jusqu’à n fois. Pour trouver toutes les adjacences d’un sommet, il faut rechercher jusqu’à n2 fois. Dans une matrice d’adjacence, il faut rechercher seulement n fois pour ces deux actions.

L’efficacité d’une matrice d’incidence dans cet algorithme augmente (relative à la matrice d'adjacence) quand le graphe a moins d'arcs et plus de sommets. Les résultats de parcourirSimple() pour graphe un et trois montrent cela :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Matrice adjacence | Matrice incidence | Opérations relatives  (incidence/adjacence)\*100 |
| Graphe un | 22 itérations | 56 itérations | (132/40)\*100 = 330% |
| 18 affectations | 76 affectations |
| Graphe trois | 145 itérations | 1158 itérations | (2076/325)\*100 = 639% |
| 180 affectations | 918 affectations |

\*Plus qu’un graphe soit connexe, plus que l’inefficacité de la matrice d’incidence augmente

En conclusion, les matrices d'incidence fournissent beaucoup plus d'informations concernant un graphe, mais pour les graphes simples ces informations sont souvent excessives et contribuent à une élévation d’affectations et itérations.