Travail à remettre

Implémentation et analyse de complexité d'algorithmes sur les graphes

Objectifs du Travail

- Rappels sur la représentation de graphes en mémoire (différentes représentations)
- Opérations sur les graphes avec calcul de complexité
- Evaluation et comparaison de complexités théoriques et expérimentales

Remise des travaux

- Le travail doit être réalisé en **monôme**
- Le rapport doit être fourni au format **pdf**. Le code source bien commenté et lisible doit être fourni au format **wordPad**
- Le code source + rapport doivent être envoyés à l'adresse works1819@gmail.com, dans deux fichiers séparés en précisant comme objet du mail « DevoirComplexité-BioInfo – nomEtudiant »
- Date limite d'envoi des travaux : samedi **06/01/2024**
- Les affichages doivent être bien alignés et lisibles.
- Les cas de copiages seront sanctionnés par un **Zéro.**

Un grand nombre de problèmes réels ont recours à la représentation de données sous forme de graphes, orientés ou non orientés. Par exemple, modéliser un réseau routier, un réseau d'ordinateurs, un automate d'états fini, un scénario de jeu à états, la planification de projets, …etc. Dans ce travail, on s'intéresse à l'implémentation des graphes en mémoire, les opérations usuelles sur les graphes et leurs complexités.

Partie I – Etablir les définitions

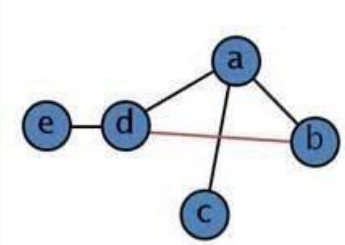
Il s'agit d'établir les principales définitions de la structure de graphe, notamment :
Graphe, orienté, non orienté, arête, arc, cycle, circuit, graphe pondéré, degré d'un nœud, arc entrant, arc sortant, ...etc.
Graphe simple, multi-graphe, graphe eulérien, arbre
Densité d'un graphe, graphe connexe, composante connexe, profondeur, graphe complet,

Partie II – Les représentations d'un graphe en mémoire

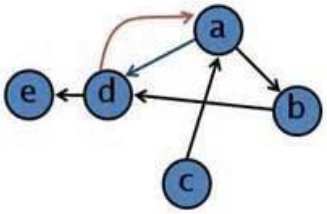
L'ensemble des nœuds est stocké dans un tableau ou une liste. Chaque nœud est décrit par ses **propriétés** qui sont : identification du nœud (lettre ou numéro), information contenue dans le nœud (selon le problème traité).
La difficulté consiste à représenter les arêtes ou les arcs (càd les liens entre les nœuds). Pour représenter ces liens, on utilise soit une représentation contigüe (**matrice d'adjacence)** soit une représentation chainée (**liste d'adjacence)**.

La matrice d'adjacence

La **matrice d'adjacence** est un tableau à deux dimensions. Chacune des dimensions est indexée parles nœuds du graphe (typiquement de 0 à N-1, sachant que N est le nombre de nœuds du graphe). A l'intersection de chaque *ligne et colonne,* on trouve un nombre : il vaut **1** si une arête relie les deux nœuds indexés par les coordonnées de la case, et **0** sinon.



	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0



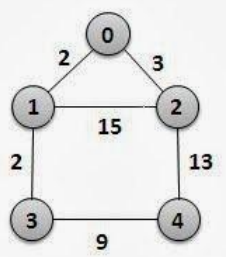
	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

On observe plusieurs caractéristiques intéressantes.

- Les boucles reliant un nœud à lui-même sont sur la diagonale de la matrice.
- Cette matrice est symétrique par rapport à sa diagonale dans un graphe non-orienté (puisque si **a** est relié à **b**, alors **b** est relié à **a**).
- Si le graphe ne comporte aucune arête, alors c'est la matrice nulle.
- Si le graphe est creux, alors la matrice le sera aussi (i.e la matrice comportera plusieurs zéros).

Remarque

Si le graphe *est pondéré*, on peut remplacer les valeurs (0 ou 1) par des nombres correspondant à la pondération (une valeur entière de poids) de chaque arête (ou arc). On fixera une valeur spéciale pour signifier l'absence d'arête (0, par exemple)



	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

Adjacency Matrix Representation of Weighted Graph

*La valeur **15** dans la case (2, 1) ou (1, 2) indique le poids de l'arc entre le nœud 2 et le nœud 1. Cela peut correspondre à une valeur de distance entre deux points par exemple.*

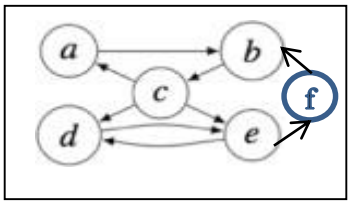
La liste d'adjacence

La **liste d'adjacence** est le moyen le plus répandu pour stocker un graphe en mémoire. La liste d'adjacence d'un nœud est la liste de ses voisins (ou la liste des arêtes qui le relie directement à ses voisins). Chaque nœud sera représenté dans un tableau avec un lien vers la liste de ses voisins.
Si le graphe est orienté, on devra différencier pour chaque nœud N, entre la liste des successeurs de N et la liste des prédécesseurs de N.

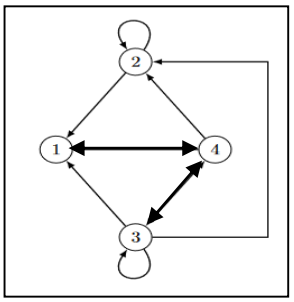
Remarque : La densité (nombre de liens) est souvent le critère décisif du choix entre la matrice d'adjacence oula liste d'adjacence.

Question

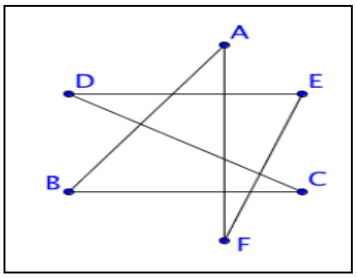
En considérant les deux représentations possibles (matrice d'adjacence, liste d'adjacence), donner la définition des deux types utilisés et montrer la représentation de chacun des graphes suivants.



Graphe G1



Graphe G2



Graphe G3

Partie III – Opérations sur les graphes et complexité

Sur chacune des deux représentations (matrice d'adjacence ou liste d'adjacence), implémenter les opérations suivantes et donner leurs complexités en dressant un tableau comparatif des différentes complexités selon les types de représentation en mémoire.

- Construction d'un graphe orienté/non orienté
- Affichage du graphe (représentation mémoire matrice d'adjacence/ liste d'adjacence). On peut également utiliser un outil graphique qui affiche le graphe à l'écran.
- Calculer la *densité* du graphe
- Calculer le *degré* du graphe
- Vérifier si le graphe est *eulérien*
- Vérifier si le graphe est *complet*
- Vérifier si le graphe est *un arbre*
- Recherche d'un *nœud* **a** dans le graphe (afficher le nœud et ses liens)
- Recherche de tous les *chemins* entre un nœud **a** et un nœud **b**
- Recherche du *chemin le plus court* entre deux nœuds **a** et **b**
- Recherche d'une *composante (fortement) connexe* à partir d'un nœud **a**.
- Trouver tous les *cycles/circuits* dans le graphe
- Ajouter/ Supprimer un nœud **a** avec ses liens
- Ajouter un lien (arc ou arête) entre deux nœuds existants

Travail demandé

1. Décrire les algorithmes (bien commentés) de chacune des opérations sus-décrites
2. Evaluer la complexité théorique de chaque algorithme en notation O de Landau, en justifiant vos calculs

3. Implémenter (en Python ou en C++), les opérations sus-décrites
4. Evaluer leurs complexités expérimentales en temps d'exécution, en augmentant la taille du graphe (nombre de nœuds : 5, 10, 20, 30, …)

***PS :** Pour les tests, prévoir plusieurs fonctions (constGraph1, constGraph2, …) qui construisent des exemples de graphes par programme (on ne va pas saisir les données au clavier).*

Indications

- L'interface principale du programme doit se présenter sous forme d'un menu à choix multiples pour répondre à l'une ou l'autre des questions de l'utilisateur.

1. Construction d'un graphe orienté (/non orienté) **2.** Affichage du graphe **3.** **4.** …

- Le programme doit être écrit sous forme modulaire (un ensemble de fonctions)

Rapport à remettre :

Un rapport doit être rédigé en suivant le plan ci-après :

1. **Introduction**
2. **Objectifs du travail**
3. **Partie I**
4. **Partie II**
5. **Partie III**

   Description des différents algorithmes (avec commentaires) et calcul de complexité

6 - Evaluation expérimentale :

En effectuant plusieurs tests du programme, remplir un tableau d'évaluation expérimentale de la complexité des algorithmes en considérant des graphes de taille 10, 20, 30, 40, 50. La densité du graphe est un facteur important pour certaines opérations.
Le Temps d'exécution (Tps) devra être évalué par le programme

Taille n	Construction (Tps)	Affichage (Tps)	Recherche d'un nœud (Tps)	…

Remarque : il est possible de présenter plusieurs tableaux selon les opérations effectuées (pour une meilleure présentation)

7. Conclusion (conclure par rapport aux complexités spatiale et temporelle dans les deux représentations, la taille et la densité du graphe)

Annexe : Joindre des captures d'écran illustrant l'exécution du programme (choisir quelques captures les plus significatives et sans redondance).