Rodrigue LEITAO—PEREIRA DIAS
Julian MAZERAT
Jessy SANFILIPPO

# TP ALG n°3

Lien github : https://github.com/Gromototo/MN.git
branche « TP ALG3 »

**- afficher_graphe_largeur(p_graphe_t g, int r) :**

On commence par initialiser une file ***q*** et on enfile le sommet de départ ***r***. Ensuite, tant que la file n'est pas vide, il défile un sommet ***s***.
Pour chaque sommet défilé, on le visite s'il n'a pas déjà été visité, en marquant son label dans un tableau de sommets visités. Ensuite, pour chaque arc sortant du sommet visité, s'il conduit à un sommet pas encore visité, ce sommet est enfilé dans la file. Le processus se répète jusqu'à ce que tous les sommets accessibles depuis le sommet de départ aient été visités.
Enfin, on affiche le label du sommet défilé.

**- afficher_graphe_profondeur(p_graphe_t g, int r) :**

On commence par initialiser une pile ***avisiter*** et on y empile le sommet de départ ***r***. Ensuite, tant que la pile n'est pas vide, on dépile un sommet ***t***.
Pour chaque sommet dépilé, on le visite s'il n'a pas déjà été visité, en marquant son label dans un tableau de visites. Ensuite, pour chaque arc sortant du sommet visité, s'il conduit à un sommet non encore visité, ce sommet est empilé dans la pile.
Le processus se répète jusqu'à ce que tous les sommets accessibles depuis le sommet de départ aient été visités en profondeur.
On affiche le label du sommet dépilé.

**- dijkstra(p_graphe_t g, int r) :**

On commence par initialiser tous les sommets du graphe avec une distance initiale de l'infini, sauf le sommet source pour lequel on initialise la distance à zéro. À chaque itération, on sélectionne le sommet non traité avec la plus petite distance temporaire (la plus petite distance estimée depuis le sommet source). On parcourt ensuite tous les voisins non traités de ce sommet sélectionné et on met à jour leur distance temporaire si elle est plus petite que la distance actuellement stockée. Une fois qu'on a traité tous les voisins du sommet sélectionné, on le marque comme traité pour ne pas le reconsidérer par la suite. On répète ces étapes jusqu'à ce que tous les sommets du graphe aient été traités. Une fois que tous les sommets ont été traités, on peut retracer le chemin le plus court depuis le sommet source jusqu'à n'importe quel autre sommet en suivant les arcs et les sommets précédemment enregistrés pendant l'exécution de l'algorithme.

**- elementaire(p_graphe_t g, chemin_t ) :**

On initialise un tableau de visites pour suivre les sommets déjà visités et un index pour suivre la dernière visite. On commence à parcourir les arcs du chemin (c.arcs). Pour chaque arc, on vérifie si le sommet de destination a déjà été visité. S'il a été visité, cela signifie que le chemin n'est pas élémentaire et il retourne 0. Sinon, il marque le sommet de destination comme visité et passe à l'arc suivant. S'il a parcouru tous les arcs sans trouver de sommet déjà visité, il retourne 1, ce qui indique que le chemin est élémentaire.

**- simple(p_graphe_t g, chemin_t ) :**

On initialise un tableau de visites pour suivre les arcs déjà visités et un index pour suivre la dernière visite. On récupère le sommet de départ (c.debut) et le sommet de destination du premier arc du chemin (c.arcs→dest→label). Pour chaque arc du chemin, On génère un identifiant unique pour l'arc en combinant les labels des sommets de début et de fin. Dans cet algorithme, l'identifiant de l'arc est calculé en concaténant le label du sommet de début avec le label du sommet de fin. On vérifie si cet identifiant a déjà été visité. Si c'est le cas, cela signifie que le chemin n'est pas simple et on retourne 0. Sinon, On marque l'identifiant de l'arc comme visité et passe à l'arc suivant. On répète ce processus pour tous les arcs du chemin. Si on a parcouru tous les arcs sans trouver d'identifiants d'arcs déjà visités, on retourne 1, ce qui indique que le chemin est simple.

**- eulerien(p_graphe_t g, chemin_t ) :**

On initialise un compteur (nb_arcs_differents) pour compter le nombre d'arcs différents utilisés dans le chemin. On initialise un tableau de visites pour suivre les arcs déjà visités et un index pour suivre la dernière visite. On récupère le sommet de départ (c.debut) et le sommet de destination du premier arc du chemin (c.arcs→dest→label). Pour chaque arc du chemin, On génère un identifiant unique de la même manière que précédemment. On vérifie si cet identifiant d'arc n'a pas déjà été visité. Si c'est le cas, cela signifie qu'un arc différent est utilisé dans le chemin, donc on incrémente le compteur (nb_arcs_differents). On passe à l'arc suivant et répète le processus. Une fois tous les arcs du chemin traités, s'il y a autant d'arcs différents utilisés que d'arcs dans le graphe total (nombre_arcs(g)), alors le chemin est eulérien et l'algorithme retourne 1. Sinon, le chemin n'est pas eulérien et l'algorithme retourne 0.

**- hamiltonien(p_graphe_t g, chemin_t ) :**

On initialise un compteur (nb_sommets_differents) pour compter le nombre de sommets différents visités dans le chemin. On initialise un tableau de visites pour suivre les sommets déjà visités et un index pour suivre la dernière visite. On récupère le sommet de départ (c.debut) et le marque comme visité. On incrémente le compteur de sommets différents. Pour chaque arc du chemin, On récupère le sommet de destination. S'il n'a pas déjà visité ce sommet de destination, On le marque comme visité et incrémente le compteur de sommets différents. S'il a déjà visité ce sommet de destination, cela signifie qu'il y a une répétition de sommets et donc le chemin n'est pas un cycle hamiltonien, il retourne 0. On passe à l'arc suivant et répète le processus. Une fois

tous les arcs du chemin traités, s'il y a autant de sommets différents visités que de sommets dans le graphe total (nombre_sommets(g)), alors le chemin forme un cycle hamiltonien et l'algorithme retourne 1. Sinon, le chemin n'est pas un cycle hamiltonien et l'algorithme retourne 0.

**- graphe_eulerien(p_graphe_t g, chemin_t ) :**

On parcourt tous les sommets du graphe. Pour chaque sommet, on vérifie si son degré est impair en utilisant la fonction (degre_entrant_sommet). Si le degré est impair, cela signifie que le sommet contribue à un nombre impair d'arcs dans le graphe. On compte le nombre de sommets ayant un degré impair. Après avoir parcouru tous les sommets, s'il y a au plus deux sommets ayant un degré impair, le graphe est eulérien. Si le nombre de sommets ayant un degré impair est inférieur ou égal à deux, l'algorithme retourne 1. Sinon, s'il y a plus de deux sommets ayant un degré impair, le graphe ne peut pas être eulérien et l'algorithme retourne 0.

**- graphe_hamiltonien(p_graphe_t g, chemin_t ) :**

cette fonction utilise la fonction récursive g_hamiltonien _rec. On commence par initialiser une structure de chemin (c) avec le label du premier sommet du graphe (g->label), aucun arc et aucun sommet. Ensuite, On appelle la fonction récursive g_hamiltonien _rec avec les paramètres suivants : Le graphe (g) ; La structure de chemin initialisée (c) ; Le premier sommet du graphe (g), qui sera le sommet de départ pour la recherche du cycle hamiltonien.

**Voici la description de la fonction g_hamiltonien_rec :**

Si la fonction hamiltonien renvoie vrai pour le chemin actuel (c) dans le graphe (g), alors un cycle hamiltonien a été trouvé, et la fonction retourne 1. Sinon, pour chaque arc sortant du sommet actuel (s), la fonction vérifie si le chemin ne passe pas par le sommet de destination de cet arc (arc_courant->dest->label). Si le chemin ne passe pas par ce sommet, cela signifie qu'il est possible d'ajouter cet arc au chemin sans former de cycle. La fonction copie ensuite le chemin actuel (c) et ajoute le nouvel arc à la copie du chemin. Elle répète ce processus récursivement pour chaque arc sortant du sommet actuel, en explorant ainsi tous les chemins possibles dans le graphe. Si l'appel récursif de g_hamiltonien_rec avec le nouveau chemin construit retourne 1 (indiquant qu'un cycle hamiltonien a été trouvé), la fonction retourne 1. Si aucun cycle hamiltonien n'a été trouvé après avoir exploré tous les arcs sortants du sommet actuel, la fonction retourne 0.

**- distance(p_graphe_t g, int x, int y) :**

Cette fonction utilise la fonction distance_rec. Elle initialise une structure de chemin (c) avec le sommet de départ (x) et aucun arc. Elle alloue dynamiquement un tableau de visites pour marquer les sommets déjà visités. Elle appelle la fonction récursive distance_rec avec les paramètres suivants : Le graphe (g) ; La structure de chemin initialisée (c) ; Le sommet de départ (chercher_sommet(g, x)) ; Le sommet de destination (y) ; Le tableau de visites ; L'index de la dernière visite.

**Voici la description de la fonction distance_rec :**

cette fonction récursive recherche toutes les possibilités de chemins entre le sommet de départ et le sommet de destination et s'il existe, renvoie le plus court.

**- excentricite(p_graphe_t g, int n) :**

La fonction initialise une variable max à zéro pour stocker la plus grande distance entre le sommet donné n et tous les autres sommets du graphe. La fonction parcourt tous les sommets du graphe. À chaque itération, elle récupère un sommet (variable s).Pour chaque sommet s, la fonction appelle la fonction distance pour calculer la distance entre le sommet n (le sommet dont l'excentricité est recherchée) et le sommet s actuellement parcouru. Si la distance calculée est supérieure à la valeur actuelle de max, la variable max est mise à jour avec cette nouvelle valeur. Ce processus est répété pour tous les sommets du graphe afin de trouver la plus grande distance entre le sommet n et tous les autres sommets. la fonction retourne alors la valeur maximale de la distance calculée.

**- diametre(p_graphe_t g) :**

Cette fonction parcourt tous les sommets du graphe et calcule leur excentricité. Elle retourne la plus grande excentricité trouvée.