﻿

Lauriane LAMOUR

Ludivine BONNET

Cylia ADOUANE

**RAPPORT PROJET ALGO**

**But du projet**

Nous avions pour projet de créer un réseau et d’établir la table de routage de chacun de ses noeuds.

**Conception du projet**

Pour créer le réseau de noeuds, nous avons décidé de créer une structure NOEUD afin de représenter un noeud du réseau. La structure NOEUD est composé d’un champ *type* donnant la classe du noeud, représentée sous la forme d’un entier (1 pour backbone, 2 pour les opérateurs de transit, 3 pour le tier 3) et d’un champ *LISTE* qui permet de sauvegarder les voisins du noeud en question.

Le champ LISTE est un pointeur sur un élément représenté par la structure *liste*. Cette structure est composée du champ *noeud* qui correspond au numéro du noeud lié au noeud auquel on veut établir ses voisins, du champ *lien* correspondant au poids du lien entre ces deux noeuds et un champ *suivant*, un pointeur sur l’élément suivant dans la liste.

Enfin pour former le réseau, nous avons conçu une structure *GRAPHE* qui contient le champ *nb\_sommets*, le nombre de sommets du graphe et un champ *tab*, un tableau de sauvegarde des noeuds.

**Initialisation du graphe**

* *GRAPHE init\_GRAPHE(int n)* : Initialisation du graphe en initialisant le tableau de

noeuds et en initialisant la liste des voisins et le type du noeud en question.

* *GRAPHE backbone(GRAPHE G)* : Création des 8 noeuds composant le backbone en choisissant aléatoirement les noeuds liés et le lien entre eux. Mise à jour de la liste des noeuds liés.
* *GRAPHE op\_transit(GRAPHE G)* : Création des 20 noeuds composant les opérateurs de transit en utilisant les fonctions :
  + *GRAPHE lien\_niv1(GRAPHE G)* : Initialisation des liens des opérateurs de transit avec le backbone en choisissant aléatoirement le nombre de noeuds du backbone auquel un opérateur de transit va être lié, choix aléatoire du/des noeuds du backbone et du lien et mise à jour de la liste de chaque noeud.
  + *GRAPHE lien\_niv2(GRAPHE G)* : Initialisation des liens des opérateurs de transit avec d’autres opérateurs de transit par le choix aléatoire du nombre d’opérateurs de transit en vérifiant que les noeuds choisis ne sont pas déjà voisin du noeud en question grâce à *int est\_voisin(GRAPHE G,int sommet,int voisin\_possible)* mais également choix aléatoire du lien. Atteinte de l’initialisation du nombre de noeuds grâce à la fonction *int nb\_noeud\_niv2(GRAPHE G,int sommet)* qui vérifie le nombre de noeuds de type 2 dans la liste des voisins du noeud en question.

﻿

* *GRAPHE op\_niv3(GRAPHE G)* : Initialisation des liens des tiers 3 avec des opérateurs de transit par le choix aléatoire des noeuds auquel un noeud de tiers 3 sera lié en vérifiant qu’ils ne sont pas déjà voisins et tirage aléatoire du poids du lien.

Initialisation des liens des tiers 3 avec d’autres tiers 3 en vérifiant au préalable le nombre de noeuds grâce à la fonction *int nb\_noeud\_niv3(GRAPHE G,int sommet)*, qui vérifie le nombre de noeuds de type 3 dans la liste des voisins du noeud en question.

**Vérification de la connexité**

La vérification de la connexité du graphe se fait à l’aide d’un marquage de coloration. Dans la fonction *int est\_connexe(GRAPHE G)*, nous avons créé un tableau d’entiers de longueur du nombre de noeuds du graphe et on initialise ce tableau à 0.  Ensuite, nous avons coder la fonction *void explorer\_sommet(GRAPHE G,int\*couleur,int sommet)* qui met le case correspondant au sommet mis en entrée à 1 pour indiquer qu’il a été visité. Elle parcourt la liste des voisins de ce sommet et applique à nouveau cette fonction. Enfin dans la fonction

*int est\_connexe(GRAPHE G)*, on effectue un parcours du tableau d’entiers et si toutes les cases sont à 1,  on renvoie 1 pour signifier que le graphe est connexe, sinon on renvoie 0.

**Etablissement de la table de routage**

Le but d’établir une table de routage est de chercher le plus court chemin entre un nœud « source » et n’importe quel autre voisin du graphe. Pour cela, nous avons décidé d’appliquer l’algorithme de Dijkstra « one to all » pour chaque nœud.

Nous avons créé une structure TABLE qui est une liste chaînée dans laquelle on stocke le sommet, son père, sa distance, ainsi qu’un pointeur vers le sommet suivant.

Pour effectuer l’algorithme de Dijkstra, nous avons mis en place 2 fonctions :

- *TABLE initialisation\_Dijkstra(GRAPHE G,int sommet)* qui ajoute les 100 éléments dans la liste TABLE à l’initialisation, si le sommet est voisin du sommet donné en paramètre, la distance sera égale à la distance entre sommet et le sommet source sinon elle sera égale à 2500 pour une distance égale à l’infini.

- *TABLE b\_p\_d(GRAPHE G, TABLE t, int source)* : dans cette fonction, nous effectuons la boucle principale de l’algorithme de Dijkstra, c’est-à-dire que nous remplissons la table issue de l’initialisation de Dijkstra.

Pour appliquer l’algorithme de Dijkstra à chaque nœud du graphe, c’est-à-dire « all to all », nous avons utilisé la fonction suivante : *TABLE\* calcul\_table\_routage\_all\_noeuds*

*(GRAPHE G)* qui renvoie un tableau de 100 structures TABLE.(algorithme Dijkstra pour 100 nœuds).